MASTER'S THESIS

Assessment of the applicability of KPIs in an education institute

Key Performance Indicators and their properties

Hilhorst, Niels

Award date:
2018

Assessment of the applicability of KPIs in an education institute

Key Performance Indicators and their properties

Programme:	Open University of the Netherlands, faculty of Management, Science & Technology Master Business Process Management & IT
Course:	IM9806 Business Process Management and IT Graduation Assignment
Student:	Niels Hilhorst
Identification number:	
Date:	31-01-2018
Version:	3.00
Status:	Final
Thesis supervisor:	Dr. ir. E.E. Roubtsova
Co-reader:	Prof. dr. ir. R.J. Kusters

Abstract

Key Performance Indicators (KPIs) are used to measure business processes that are focused on the achievement of a set of business goals. The educational sector set new business goals to respond to current challenges in the sector. KPIs should correspond to these challenges. One of the current challenges is *registering absent students and preventing student dropouts.*
It is unclear which KPIs correspond with this challenge and whether the KPIs are 'well-defined'.
This study focuses on what a 'well-defined' KPI comprises and identifies methods to validate them.
Based on definitions found in literature, a conceptual model of the notion of a KPI is constructed. This conceptual model shows what should be modeled or implemented to calculate and validate KPIs. The conceptual model is used to choose a method that covers all the concepts needed for KPI definition and validation. The chosen method will be used to validate designed KPIs on the case study of *registering absent students and preventing student dropouts.*

Keywords

KPIs, properties, validation, applicability, education.

Contents

1. Introduction

Education is focused on teaching individuals to function properly in a society. It is a crucial sector of society defining its prosperity in the future and therefore has a social responsibility.
The educational sector continually undergoes changes caused by technological-, demographic-, economic- or legislation changes. Education organizations set new business goals to respond on these challenges.
Key Performance Indicators (KPIs) are used to measure business processes that are focused on the achievement of the set business goals based on current challenges. KPIs should correspond with these challenges. The question is which KPIs correspond to the current challenges?
Performance measurement can indicate a lot of different things like student satisfaction, study success, occupation of teachers, total number of students or total number of student dropouts. Other sorts of measurements may indicate student presence and absence registration information. These indicators, for example, concern the presence of students during mandatory classes. Another example of performance measurement is absenteeism of teachers and students.

2. Research problem

Practice shows that presence of students in classes is a key condition for successful education. Therefore, one of the education processes concerns registering absent students and preventing student dropouts.
Organizations use performance measures in the form of KPIs. It is however unclear what KPIs correspond with the current challenge of registering absent students and preventing student dropouts and whether these KPIs are 'well-defined' to react on this challenge.

2.1. Research questions

The main research question (MRQ) is:

<u>What 'well-defined' KPIs correspond to the current educational sector challenge of registering absent students and preventing student dropouts?</u>

To answer the MRQ, six sub-research questions (SRQs) are formulated. Aggregating the answers of the SRQs answer the MRQ. This study is conducted in two phases, the literature- and the empirical-phase. SRQs one to four are part of the literature-phase. SRQs five and six are part of the empirical-phase.

The six SRQs outlined per phase are:

<u>Literature-phase:</u>
SRQ1: *What concepts define a KPI?*
SRQ2: *What properties define the definition of a 'well-defined' KPI?*
SRQ3: *What is the relation between the concepts from definitions and properties?*
SRQ4: *What methods exists to validate KPIs?*

<u>Empirical-phase:</u>
Case study: Modeling of the process and the KPIs of registering absent students and preventing student dropouts in a senior vocational education institute in the Netherlands.

SRQ5: *Do the KPIs designed for the case possess the properties of a 'well-defined' KPI?*
SRQ6: *How to evaluate the method of KPI design with design science?*

Table 1 outlines which research methods will be used per SRQ. For the literature-phase, a critical literature review is accompanied with conceptual modeling based on definitions found in literature. A conceptual model of the notion of KPI will be constructed.

For the empirical-phase, desk research and modeling are combined as new KPIs may be designed and validated. It is a case study of modeling and the use of the design science analysis (via guidelines) of the method for designed KPIs.

Table 1: Research methods per SRQ

Phase	SRQ #	Research method
Literature	1	Literature review
	2	Literature review
	3	Literature review / conceptual modeling
	4	Literature review
Empirical	5	Desk research / modeling
	6	Desk research / modeling

A research model helps to visualize the distinct phases a study passes (Verschuren & Doorewaard, 2007). Figure 1 shows the visualization of this study:

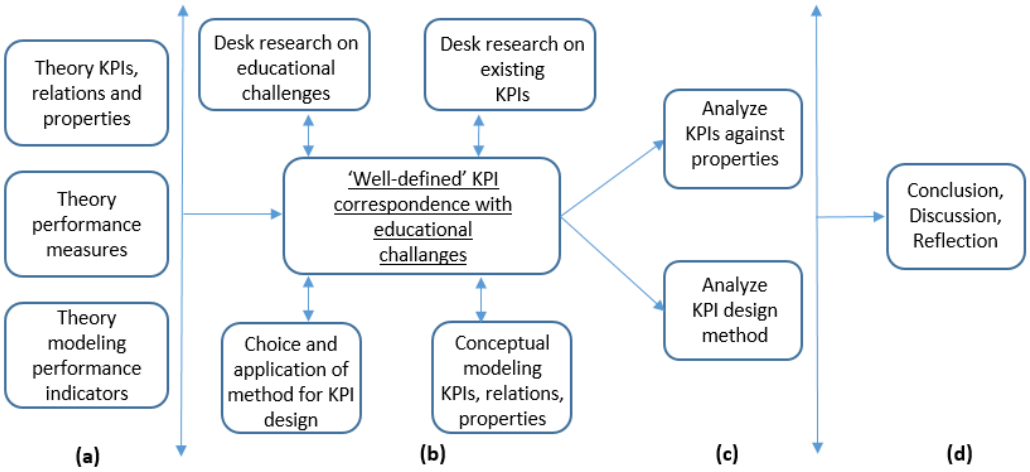


Figure 1: Research model of this study

The literature study (a) forms the theoretical framework and forms the basis for (b) which is empirical based. In (c) the empirically gathered data is analyzed and in (d) the conclusion, discussion and reflection of the study will be formed based on (a), (b) and (c).

3. SRQ1: *What concepts define a KPI?*

The goal of this section is to collect the definitions of a KPI and relate all concepts used in different definitions of KPIs. The collection of definitions is done by a review of a restricted set of often cited scientific papers. The definitions have been collected for further analysis by conceptual modeling.

Eckerson (2009) states that "*A KPI is a PI that embodies a strategic objective and measures performance against a goal*". In this definition three concepts are found, namely: *KPI*, *strategic objective* and *goal*. To use these three concepts in the conceptual model, it is desirable to find other definitions to confirm these concepts. Confirmation means that others mention the same concepts.

Roubtsova & Michell (2014) state the following: "*A KPI is a cumulative function of the cardinality of a set of selected business objects (presented by protocol machines) and the values of their attributes*". This definition also contains concepts, namely: *KPI*, *business objects* and *attributes*. This definition also states that a KPI is a function and that attributes can have values.

Eckerson (2009) and Roubtsova & Michell (2014) combined contain the following five concepts (1) *KPI,* (2) *strategic objective,* (3) *goal,* (4) *business objects* and (5) *attributes.*
In the used literature, not always a clear definition exist that explicitly define a concept. Instead many citations found in the literature describe the relation between the concepts. Therefore, it is possible to implicitly derive the concepts from the definitions describing the relation between the concepts. It also makes it possible to confirm explicitly found concepts (in definitions) and therefore this 'technique' is chosen to identify new or confirm already identified concepts.

Eckerson (2009) mention seven attributes that can assist users in interpreting KPIs and state that *"These seven attributes combine to provide valuable insight into the state of performance".* Roubtsova & Michell (2014) state in relation to the concept *business objects* that they have attributes and that the attributes can have values. Based on Eckerson (2009) and Roubtsova & Michell (2014) it is concluded that attributes have a relation with KPIs and that the (5) *attribute* concept can be confirmed to be used in the conceptual model.

Other citations about KPIs were found in the literature that help complete the list of concepts that define a KPI or to confirm concepts already identified. For example, Maté (2017) states that: *"KPIs monitor goals, processes, or situations".* In this statement the concepts *processes* and *situations* are 'new' concepts whereas the concept *goals* is already identified. Kueng (2000) states that: "*Performance indicators are derived either from business process goals or from the means of achieving the goals".* A recommendation from Neely (1997) is that: "*Performance measures should relate to specific goals (targets)"*. Combining the statements of Mate (2017), Kueng (2000), and Neely (1997), it is concluded that the concept (3) *goals* can be confirmed. Processes and situations cannot be confirmed yet.

To confirm the concept processes derived from Maté (2017), Lohman et.al (2004), Kueng (2000), and Neely (1997) can be used. Lohman et.al (2004) is about designing a performance measurement system (PMS) and this closely relates to KPIs. It states that: "*A performance indicator (PI) is a variable that expresses quantitatively the effectiveness or efficiency or both, of a part of or a whole process, or system, against a given norm or target"*. Like Lohman et.al (2004), Kueng (2000) is about a PMS, it states that: "*The measurement system should be focused on processes, not on whole organizations or organizational units"*.

A recommendation from Neely (1997) is that "*Performance measures should reflect the "business process" – i.e. both the supplier and customer should be involved in the definition of the measure".* With Lohman et.al (2004), Kueng (2000), and Neely (1997) we can conclude that *processes* is a confirmed concept that help define a KPI and will be identified as concept (6). *Situations* cannot be confirmed and will not be used in this study as they used to describe processes.

Neely (1997) mention a recommendation based on eight various sources that: "*Performance measures should be derived from strategy*". A statement of Lohman et.al (2004) is that: "*PM is an activity that managers perform in order to reach predefined goals that are derived from the company's strategic objectives*". Performance measurement (PM) has a close relation to KPIs therefore, this statement will be used. Neely (1997) and Lohman et.al (2004) are used to confirm the concept (2) *strategy.*

It is concluded that every found concept in the literature can be confirmed via various sources except the concept *business objects*. Although this concept cannot be confirmed it can be used to define what a KPI is and be added to the conceptual model. Defining business objects help to calculate KPIs. Also, because business objects exist 'in' processes and the processes concept can be confirmed it is justified to use business objects as a concept in the conceptual model.

Although a KPI can be defined as a concept based on the literature, in this study a KPI is not seen as a concept on its own. The reason is twofold: first, because a KPI is more a formula than a concept which is supported by Roubtsova & Michell (2014) in the KPI definition, and second, the other concepts together define 'what' a KPI is. All the identified concepts together form a KPI and can only exist if the other concepts exist. A KPI will still be added as a concept to the conceptual model. The reason is because it helps to get a clear overview (by showing the relations with other concepts) and to get a complete model. However, it is important to understand that the other concepts 'form' the KPI.

The six concepts in the various definitions found in the literature give valuable insight about what concepts define a KPI. The concepts are variables that can have different values. It also gives insight in the mutually relation between the different concepts. In chapter 3.1 the concepts and their relation will be used to build a conceptual model that can be used to calculate KPIs.
However, to calculate KPIs a concept *time* is needed because measures on a particular time are 'snapshots' in time and it is impossible to calculate with only one 'snapshot'. It is necessary to have more than one 'snapshot' in time for calculation. Roubtsova & Michell (2014) state the following: *"Key Performance Indicators are cumulative measures of system achievements during a given time period"*.
To address the time period the concept *time* will be added to the list of concepts that define a KPI.

It is concluded that seven concepts can be identified that define a KPI. Because the context of KPIs is 'business' this term will be added to three of the seven concepts to make the context clear. The term will be added to the concepts: *strategy*, *goals*, and *processes*.

In figure 2 the identified concepts are used to build the conceptual model. In chapter 3.1 the relation between the concepts are identified and added to the model.

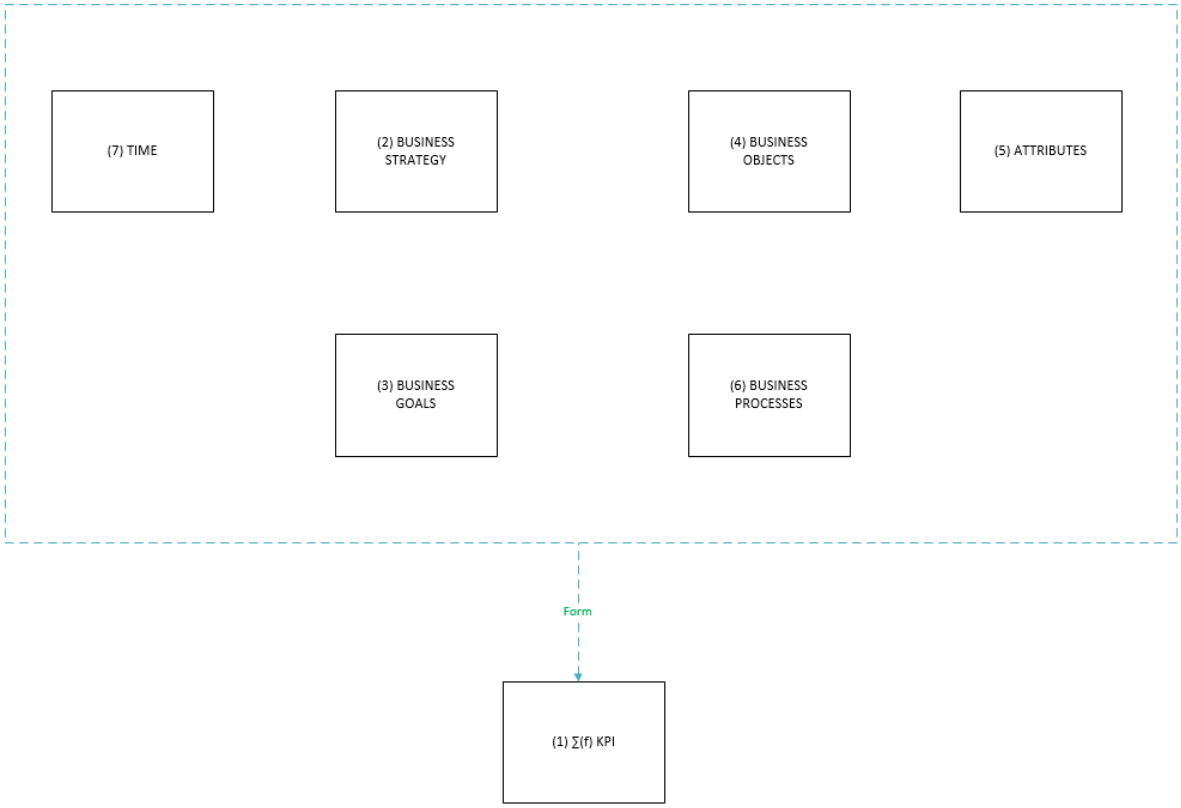


Figure 2: Conceptual model of concepts

An overview of the seven concepts are summarized and presented in table 2 each with a description of the concept:

Table 2: Concepts that define a KPI

#	Concept	Source	Description
1	KPI	(Eckerson, 2009) (Roubtsova & Michell, 2014)	*"A KPI is a PI that embodies a strategic objective and measures performance against a goal"* (Eckerson, 2009); *"A KPI is a cumulative function of the cardinality of a set of selected business objects (presented by protocol machines) and the values of their attributes"* (Roubtsova & Michell, 2014); *"KPIs use a metric for quantitatively assessing performance regarding the needs and expectations of stakeholders, the achievement of goals, and reflecting the critical success factors"* (Sinclair & Zairi, 1995).
2	Business strategy	(Neely, 1997) (Lohman et.al., 2004)	*"Broad formula for how a business is going to compete, what its goals should be, and what policies will be needed to carry out those goals."* (Porter, 1980); *"The determination of the basic long-term goals and objectives of an enterprise, and the adoption of courses of action and the allocation of resources for carrying out these goals."* (Chandler, 1962).

Continued table 3: Concepts that define a KPI

#	Concept	Source	Description
3	Business goals	(Maté, 2017) (Kueng, 2000) (Neely, 1997)	*"A goal is an organizational objective"* (Popova & Sharpanskykh, 2010).
4	Business objects	(Roubtsova & Michell, 2014)	These are objects that can be derived from business processes and can be used for calculation i.e. an object can be a *student*.
5	Attributes	(Roubtsova & Michell, 2014) (Eckerson, 2009)	Attributes can have values that belong to a concept and can be used for calculation.
6	Business processes	(Maté, 2017) (Lohman et.al.,2004) (Kueng, 2000) (Neely, 1997)	*"Business processes are responsible for the realization of tactical goals."* (Maté, 2017).
7	Time	(Roubtsova & Michell, 2014)	*"Key Performance Indicators are cumulative measures of system achievements during a given time period."* (Roubtsova & Michell, 2014).

3.1. The relation between the concepts

In chapter 3 concepts are identified using definitions of KPIs. Based on the definitions it is concluded that the following relations exist between the concepts:

- The concept "KPI" and the concept "business objects" are related with the relation: *"KPIs represent a set of business objects"*
- The concept "KPI" and the concept "time" are related with the relation: *"KPIs measure performance during a given period of time"*
- The concept "KPI" and the concept "business goals" are related with the relation: *"KPI measure performance against a business goal"*
- The concept "KPI" and the concept "business strategy" are related with the relation: *"KPI is a degree of achievement of a business strategy"*
- The concept "business goals" and the concept "business strategy" are related with the relation: *"Business goals are translated into business strategy"*
- The concept "business objects" and the concept "business processes" are related with the relation: *"Business objects come from business processes"*
- The concept "business objects" and the concept "attributes" are related with the relation: *"Business objects have attributes"*
- The concept "business processes" and the concept "business goals" are related with the relation: *"Business processes realize business goals"*
- All concepts and the concept "KPI" are related with the relation: *"All concepts form a KPI"*.

The concepts that define a KPI and the relation between the concepts are used to further build the conceptual model, see figure 3. It shows that a method for KPI design should model *business processes*, *business goals* and *business objects* with *attributes* at given *time* intervals.

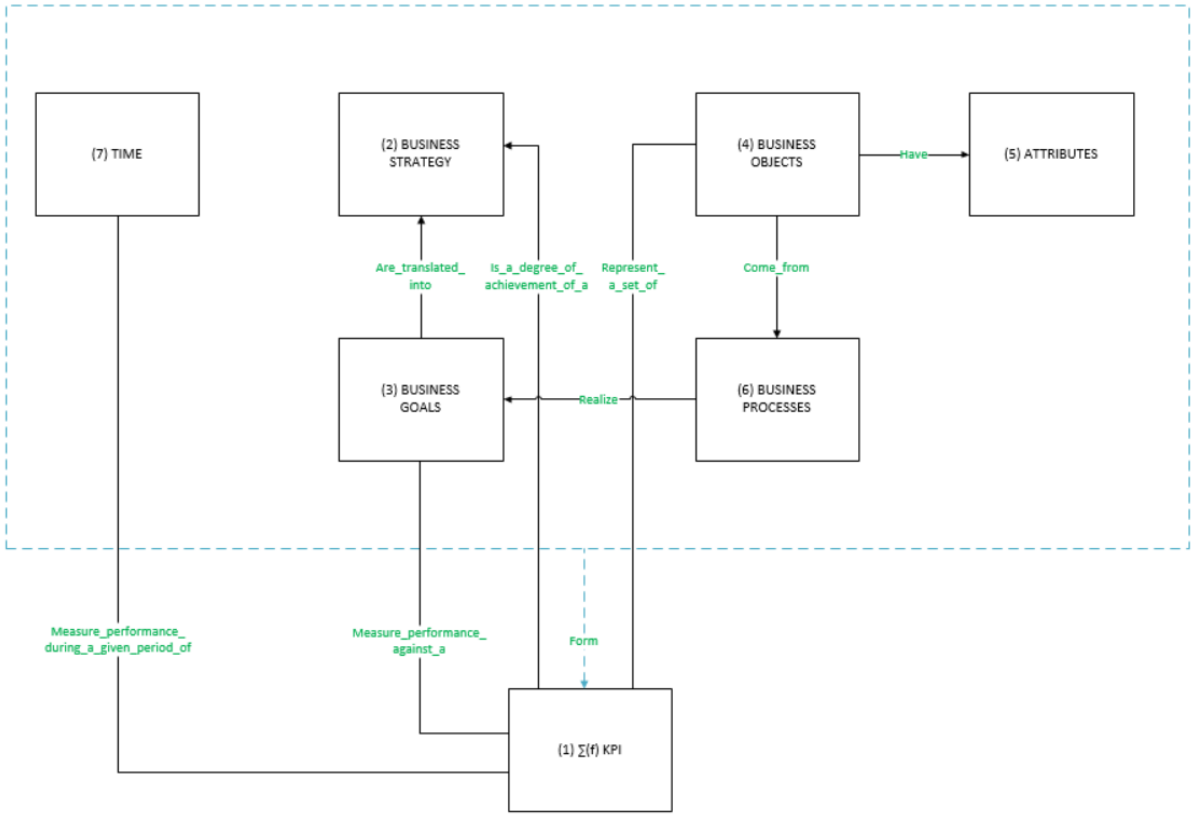


Figure 3: Conceptual model of concepts and relations

4. SRQ2: *What properties define the definition of a 'well-defined' KPI?*

The conceptual model in chapter 3 shows only the concepts needed for calculation of KPIs. As the KPIs are interpreted by stakeholders' participation in the business process, KPIs can change the behavior of stakeholders. That is why the properties showing the influence of KPIs on the stakeholders are usually used in the design of KPIs.

The goal of this section is to collect the properties that define a 'well-defined' KPI. The properties are needed for calculation and validation of KPIs. The collection of properties is done by a review of the same restricted set of often cited scientific papers used in SRQ1.

The following sources will be used: Neely (1997), Eckerson (2009), Kueng (2000), and Roubtsova & Michell (2014). Because these various sources do not use the same term, for clarification reasons this study will use the term 'properties'. This term relates to the recommendations, characteristics and requirements of respectively Neely (1997), Eckerson (2009), and Kueng (2000). Roubtsova & Michell (2014) already use the term properties.

Neely (1997) studies the design of performance measures and identifies 22 recommendations that provide an indication of the characteristics a 'well-designed' performance measure exhibits. The 22 recommendations are grouped in 10 elements on a performance record sheet.
Neely (1997) draws on the experiences gained during five specific applications of the record sheet in the aerospace and automotive sectors and concludes that 13 of the 22 recommendations satisfy, see table 3. Satisfy means that these recommendations are exhibited by the five applications.

Table 3: 13 recommendations from Neely (1997)

Recommendation		
1) "Performance measures should be derived from strategy"	*6) "Performance measures should be relevant"*	*11) "Performance measures should be based on an explicitly defined formula and source of data"*
2) "Performance measures should be simple to understand"	*7) "Performance measures should be part of a closed management loop"*	*12) "Performance measures should provide information"*
3) "Performance measures should provide timely and accurate feedback"	*8) "Performance measures should be clearly defined"*	*13) "Performance measures should be precise – be exact about what is being measured"*
4) "Performance measures should be based on quantities that can be influenced, or controlled, by the user alone or in co-operation with others"	*9) "Performance measures should provide fast feedback"*	
5) "Performance measures should relate to specific goals (targets)"	*10) "Performance measures should have an explicit purpose"*	

Eckerson (2009) identifies ten characteristics of KPIs that are based on guidelines to create effective KPIs, see table 4.

Table 4: Ten characteristics of Eckerson (2009)

Characteristic	**Description**	**Characteristic**	**Description**
1) Sparse	*"The fewer KPIs, the better"*	6) Referenced	*"Users can view origins and context"*
2) Drillable	*"Users can drill into detail"*	7) Correlated	*"KPIs drive desired outcomes"*
3) Simple	*"Users understand the KPIs"*	8) Balanced	*"KPIs consist of both financial and non-financial metrics"*
4) Actionable	*"Users know how to affect outcomes"*	9) Aligned	*"KPIs don't undermine each other"*
5) Owned	*"KPIs have an owner"*	10) Validated	*"Workers can't circumvent the KPIs"*

Kueng (2000) mention six requirements which performance indicators should fulfill based on the work of Kitchenham (1996) and Winchell (1996). The information from Kueng (2000) is presented in tabular form for clarification reasons, see table 5.

Table 5: Six requirements of Kueng (2000)

Requirement	**Description**
1) Quantifiability	*"If performance indicators are not quantitative by nature, they have to be transformed"*
2) Sensitivity	*"Sensitivity expresses how much the performance must change before the change can be detected"*
3) Linearity	*"Linearity indicates the extent to which process performance changes are congruent with the value of a certain indicator"*
4) Reliability	*"A reliable performance indicator is free of measurement errors. To illustrate, if a certain business process has to be rated through a given performance indicator by different experts, the results should not depend on the subjective evaluation of an Individual"*

Continued table 5: Six requirements of Kueng (2000)

Requirement	Description
5) Efficiency	*"Since the measurement itself requires human, financial and physical resources it must be worth the effort from a cost /benefit point of view"*
6) Improvement-oriented	*"Performance indicators should emphasize improvement rather than conformity with instructions. Therefore, measuring billing errors, number of safety violations, data entry errors and the like do not create an atmosphere where feedback sessions are viewed in a positive, constructive light"*

Roubtsova & Michell (2014) have chosen a set of five properties proposed in Kueng (2000) and Parmenter (2010). Roubtsova & Michell (2014) formalize them based on their definitions. These properties will also be presented in tabular form for clarification reasons. See table 6.

Table 6: Definitions of properties from Roubtsova & Michell (2014)

Property	Definition	Description
1) Quantifiable	*A KPI should be in a quantifiable form*	*"Quantification is an act of selection and counting."*
2) Sensitive	*A KPI needs to be sensitive to changes*	*"The sensitivity is the minimum magnitude of the change of the sensed element of the model, required to produce a noticeable value of a KPI."*
3) Linear	*A KPI should be linear*	*A KPI formula can have possible arguments. The function can make the KPI function linear for this argument (A linear mathematical relationship)*
4) Semantically reliable	*A KPI should be semantically reliable*	*Measuring the semantic reliability of a KPI is done by the number of additional assumptions that need to be made to derive the KPI value. If the number is '0' the KPI is semantically reliable*
5) Oriented to improvement	*"A KPI should be oriented to improvement, not to conformance to plans."*	*The improvement orientation of a KPI is measured by the number of manipulative scenarios. If the set of manipulative scenarios is empty the KPI is oriented to improvement*

The various sources mention quantitative or qualitative properties. Quantitative properties can be calculated and via calculation be validated. Validation is determining if a property of a KPI is satisfied.

Qualitative is subjective and relates to properties in a user perspective and cannot be easily calculated. These user perception properties use an extended set of concepts and need other methods for validations than calculation. For example, recommendation two of Neely (1997) states that: *"Performance measures should be simple to understand".* The word '*simple'* is a subjective and qualitative term and therefore difficult to measure. It is possible to make subjective terms like 'simple' quantitative by using surveys, but surveys need a lot of respondents and take a lot of time. Because of time constraints and the lack of access to respondents this study will not use surveys and therefore the focus lies on quantification.

The *recommendations* (Neely, 1997), *characteristics* (Eckerson, 2000), *requirements* (Kueng, 2000) and *properties* (Roubtsova & Michell, 2014) will be used to collect all properties that are quantitative.

The result of the identified quantitative properties of table 3, 4, 5, and 6, are summarized in table 7 in alphabetical order.

Table 7: Quantifiable properties of a 'well-defined' KPI

#	Property
1	Improvement-oriented
2	Linear
3	Quantifiable
4	Reliable
5	Sensitive

Improvement-oriented means that performance indicators should emphasize improvement rather than conformity with instructions (Kueng, 2000). Roubtsova & Michell (2014) state about this property that a KPI should be oriented to improvement, not to conformance to plans which is called a plan-oriented KPI. An improvement-oriented KPI corresponds to a business goal and guarantees that the arguments of the KPI formulas are objectively changed by the business process and cannot be manipulated. A plan-oriented KPI (value) may be achieved through manipulating of numbers of instances in the business process and this is not desirable.
This property can be validated by analysis of the scenarios of *business processes,* execution of *business processes* and comparison of the execution results with *business goals* during given *time periods*. In chapter 5 is described why this is concluded. This also applies for the following properties.

Linear indicates the extent to which process performance changes are congruent with the value of a certain indicator (Kueng, 2000). Roubtsova & Michell (2014) state that a KPI must be linear and this simplifies decision making.
This property can be validated via the concepts: *business processes* and *business objects*.

Quantifiable is an act of selection and counting (Roubtsova & Michell, 2014). Kueng (2000) states that if performance indicators are not quantitative by nature, they should be transformed.
This property can be validated via the concepts: *business objects*, *attributes*, and *time*.

The property *reliable* means that a reliable performance indicator is free of measurement errors (Kueng, 2000). Roubtsova & Michell (2014) state that a KPI must be semantically reliable and that the set of additional assumptions must be empty.
This property can be validated via the concept: *business process*. Guaranties of KPI reliability often demands extension of business processes to cover the missing assumptions.

Sensitive expresses how much the performance must change before the change can be detected (Kueng, 2000). Roubtsova & Michell (2014) state about this property that a KPI should be sensitive to changes. *"The sensitivity is the minimum magnitude of the change of the sensed element of the model, required to produce a noticeable value of a KPI."* (Roubtsova & Michell, 2014).
This property can be validated via the concepts: *business process*, *business objects*, and *attributes*.

The properties in table 7 are selected as properties that define the definition of a 'well-defined' KPI. Chapter 5 describes how is determined which concepts can help validating the properties and these properties are then added to the conceptual model to get a complete model.

5. SRQ3: *What is the relation between the concepts from definitions and properties?*

The goal of this section is to further build the conceptual model based on the identified concepts and properties described in chapter 3 and 4. In chapter 5 the relation between the concepts and properties is determined and the conceptual model is further extended. The properties will be placed next to the concepts that allows validation of the property. Chapter 5 shows a complete model with concepts, the relation between concepts and their associated properties. The complete model gives the basis for a common understanding and specification of a KPI in general and can be used for calculation and validation of (existing) KPIs.

The placement of the identified properties in the conceptual model is based on citations of Kueng (2000) and Roubtsova & Michell (2014).

Table 8: Citations of properties by Kueng (2000), and Roubtsova & Michel (2014)

Citations about property	**Kueng (2000)**	**Roubtsova & Michell (2014)**
Improvement-oriented	*"Performance indicators should emphasize improvement rather than conformity with instructions."*	*"For validation of this property on a model, the improvement of the system should be defined from the definition of system goals."* *"The improvement should be related to changes of a KPI in sequential time intervals."* *"The model of the system and the system itself should guarantee that the arguments of the KPI formulas are objectively changed by the business process and cannot be manipulated."*
Linear	*"Linearity indicates the extent to which process performance changes are congruent with the value of a certain indicator. Or, conversely, a small change in the business process performance should lead to a small change in the value of a corresponding performance indicator, whereas an ample performance rise should also lead to strong change in the level of the performance indicator."*	*"The linearity of a KPI can be tested using the executable model for each KPI. For testing, the model should be populated with objects."*
Quantifiable	*"If performance indicators are not quantitative by nature, they have to be transformed. For instance, the performance indicator customer payment attitude could be transformed into number of days between `invoice sent' and `invoice paid' ."*	*"Quantification is an act of selection and counting. Our definition of a KPI shows that selection and counting of instances of business objects (protocol machines) of a particular sort is the way of quantification. The result of selection and counting is used as an argument of the function for calculation of a KPI or for the access to the attribute values of selected objects."* *"It contains a predicate for the selection of a number of business objects (presented as protocol machines) using the given time interval and a set of given values of object attributes."*

Continued table 8: Citations of properties by Kueng (2000), and Roubtsova & Michel (2014)

Citations about property	Kueng (2000)	Roubtsova & Michell (2014)
Reliable	*"A reliable performance indicator is free of measurement errors. To illustrate, if a certain business process has to be rated through a given performance indicator by different experts, the results should not depend on the subjective evaluation of an individual."*	*"From the modelling perspective, we measure the semantic reliability of a KPI by the number of additional assumptions that need to be made in order to derive the KPI value. If the set of additional assumptions is empty, the KPI is semantically reliable."*
Sensitive	*"Sensitivity expresses how much the performance must change before the change can be detected. Therefore, a sensitive indicator is able to detect even minor changes in performance."*	*"Our model shows that the argument of the function can fall into two categories: a number of business objects or a variable that depends on values of attributes of the group of selected objects."* *"The sensitivity is the minimum magnitude of the change of the sensed element of the model, required to produce a noticeable value of a KPI."*

The concepts mentioned in the citations in table 8 are highlighted in red and are used to decide where the properties should be placed in the conceptual model. In the citation from Kueng (2000) about the property sensitive, *performance* is highlighted because this relates to the concept *business process*.

The following is concluded:

- The property *improvement-oriented* should be placed by the concepts *time, business goals,* and *business processes*
- The property *linear* should be placed by the concepts *business objects,* and *business processes*
- The property *quantifiable* should be placed by the concepts *time, business objects,* and *attributes*
- The property *reliable* should be placed by the concept *business processes*
- The property *sensitive* should be placed by the concepts *business objects*, *attributes,* and *business processes.*

In figure 4 the properties are added to the conceptual model to get a complete model. This model gives the basis for a common understanding and specification of a KPI in general and can be used for calculation and validation of (existing) KPIs.

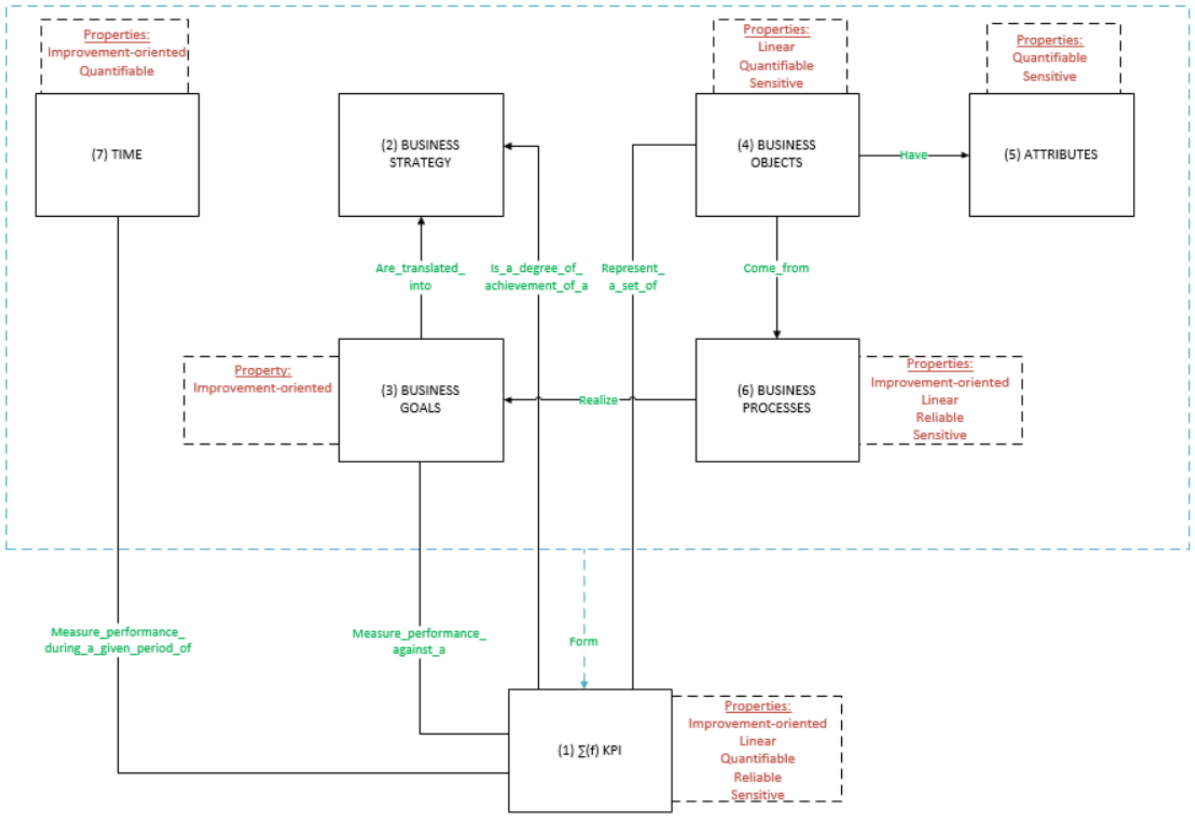


Figure 4: Conceptual model of concepts with associated properties

By looking at the model (figure 4) it becomes clear that no property is placed next to the concept business strategy. This means that this concept cannot be directly used for property validation. Lohman et.al (2004) states that performance measurement is an activity that managers perform to reach predefined goals that are derived from the company's strategic objectives. So, strategy can be seen as a high-level plan to achieve one or more business goals. But KPIs monitor the delivery of strategy and that is why the KPI is associated with the business strategy concept as can been seen in figure 4 with the *degree of achievement* relation between a KPI and a business strategy.

A business strategy can set a norm or target for a measure. Eckerson (2009) states that it is key that this norm or target for the measure is derived from business strategy. Without a norm a valid indication of the performance cannot be done. A KPI needs to be measured against a norm derived from business strategy. A norm can be for example that 80% percent of absent students must have mentor support. The value of a KPI can then be compared with the norm. Only then it is possible to determine if the performance is higher or lower than the norm.

6. SRQ4: *What methods exists to validate KPIs?*

The goal of this section is to describe existing methods that can be used to validate KPIs using the identified concepts, relations and their properties.

Conceptual modeling is the activity of formally describing some aspects of the physical and social world around us for understanding and communication (Mylopoulos, 2009).
KPIs are numerical functions, using the concepts as variables. Properties are functions, that have values “true, false” or numeric values. In chapter 5 a complete model with concepts, the relation between the concepts and the associated properties is shown.

This conceptual model can be used for calculations by showing what information (in a business case) is needed to do this. For example, a description of a business processes, its business objects (with their attributes), and business goals.

Most of the existing methods that focus on performance management agree that KPIs have required properties, but the existing methods use modeling semantics that cannot guarantee that the designed KPI will poses the required properties. The KPIs being implemented run the risk of being unreliable or plan-oriented.

Roubtsova & Michell (2014) state that a method for validation should enable simple and easy changeable executable models. This is needed to correct assumptions about the business process used for KPI definitions. Roubtsova & Michell (2014) found that existing methods based on the work of Popova & Sharpanskykh (2010), Strecker et. al. (2012), and Letier et. al. (2008) lack the support of easy changeable executable process models needed for validation of properties of KPIs. It is assumed that this is because of semantic incompatibility of the goal modeling- and conventional process modeling approaches. Using these methods may potentially introduce unreliable or plan-oriented KPIs.

The ExtREME-method designed and proposed by Roubtsova & Michell (2014) exploits the semantic compatibility of the goal modelling and protocol modelling approaches. It allows for (easily) changing the executable model and enables early validation of KPI properties on a business process used for KPI definition. This eliminates incompleteness of assumptions about the business process and can prevent implementation and application of unreliable or plan-oriented KPIs in organizations.

7. Conclusion literature phase

Seven concepts are identified that define a KPI. These are *business strategy, business goals, business objects, attributes, business processes* and *time*. The seventh concept is the KPI itself and is 'formed' by the other six concepts. The concepts are all related to each other and nine relations are identified between them. The identified concepts and relations are used to design a conceptual model. The conceptual model identifies what information in a (business) case study is needed to be able to validate KPIs. For example, a description of a business case (concept business processes), its business objects (concept business objects) with their attributes (concept attributes), the business goals (concept business goals) and the rest of the concepts in the model.

The characteristics of the ExtREME-method described in chapter 6 allows for validation of the five properties that define the definition of a 'well-defined' KPI. These are *improvement-oriented*, *linear, quantifiable*, *reliable*, and *sensitive*. This method proves the ability to validate these properties via calculation of (the value of) concepts and their attributes. The properties that ExtREME promise to validate are placed in a conceptual model to get a complete model that gives the basis for a common understanding and specification of a KPI in general and can be used for calculation and validation of (existing) KPIs.

8. Case study of ROCvA: *"Notice and report presence and absence"*

KPIs are complex because they combine several aspects like a *business process*, *behavior of people*, and *interaction with technology*. To explore such complex research objects, a case study method is often used. A case study explores a research topic or phenomenon within its context, or within a number of real-life context (Yin, 2009). The case study is relevant to gain a rich understanding of the context of the research and the process being enacted (Morris & Wood, 1991). However, case studies do not generate the same results on e.g. causal relationships as controlled experiments do, but they provide deeper understanding of the phenomena under study (Runeson & Höst, 2008).

The case study is directed to design KPIs for presence registration at an education institute and find if the conceptual model (from the literature phase) and the ExtREME-method allows to design KPIs and identify if there are scenarios where some properties of KPIs are not met.

8.1 Business process: *"Notice and report presence and absence"* and business goals for KPIs

Senior vocational education (SVE) is one of the Dutch educational sector segments. In the Netherlands, this is referred to as middelbaar beroeps onderwijs (MBO). The case organization operates in this sector and facilitates educational services for more than 33000 students and employs more than 2000 faculty members.

The description of the process used in this research has been taken from the educational sector triple A architecture framework (SAMBO-ICT, 2012). This framework is used as a starting point for all designs and developments in the field of business-, information- and technical architecture. The framework offers a functional design for processes and it describes business processes to achieve a standardized way of performing business/education and is used by a lot of other SVE/MBO-institutes in the Netherlands. The business processes of the case organization are based on this framework.

This case study focusses on the process *"Notice and report presence and absence"* which is a subprocess of "*Analyzing presence and absence*". In Dutch the proces name is *"Analyseren aan- en afwezigheid"*, see figure 5.

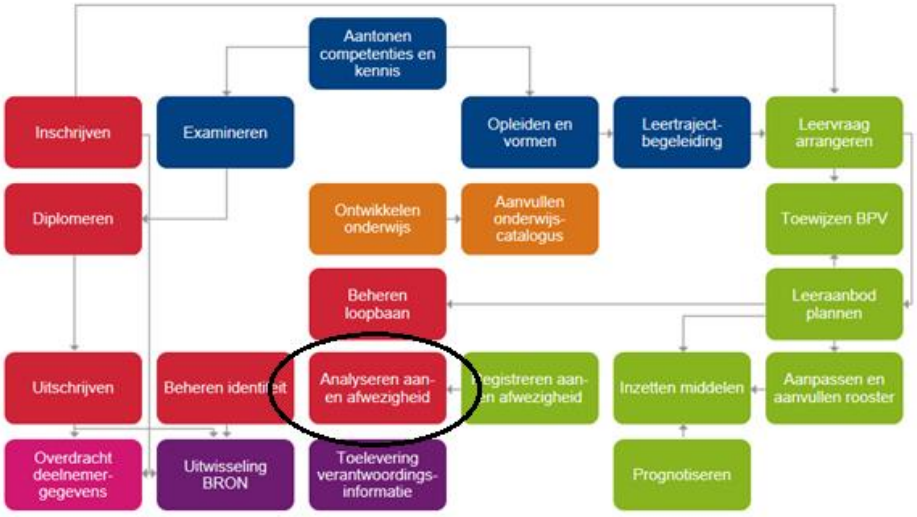


Figure 5: Educational business processes (SAMBO-ICT, 2012)

The case organization has a system that contains the registration of students who are absent and for how long. Students that are absent for more than two weeks need to be reported to an attendance officer.

A description of the chosen business process that will be used as a basis for KPI design is presented below:

A teacher registers in the begin of every class which students are not present.

Everyday an absentee worker checks which students are not present for more than two weeks and must mark a student absent. An absentee worker must report an absent student to the attendance officer, this must be done within 5 days.

The absentee worker informs the absent student about his report to the attendance officer. If the student is under the age of eighteen the student is informed as well as the parents of the student. If the student is older than eighteen the absentee worker informs only the student.

After reporting the student an absentee worker arranges a mentor for a student. A mentor gives support to an absent student. Support means contacting the student, making agreements and motivate the student to return.

An administration worker marks a student as dropout if a student doesn't return in 60 days or marks a student as returned.

The in this section described organization and business process are used to design KPIs following the ExtREME-method. In a later stage of this study this allows for evaluation of KPI design and evaluation of the results. In the next section the goals, requirements and goals of measurements of the process will be identified.

8.2 ExtREME modeling of the goals and the business process *"Notice and report presence and absence"*

The modeling techniques of the ExtREME-method are used to design a model of goals and an executable (protocol)model of the process. Also for design of KPIs, simulation of the model (and the KPIs) and validation of the KPIs against goals and against properties. Specifically, searching for the evidence of (non-)reliability and (plan) improvement orientation of KPIs and on other properties of KPIs. Next the ways of communication with stakeholders based on the KPIs model is proposed.

8.2.1 Goal model, goals of measurement, requirements

A goal model contains the goals of the underlying business process and goals of measurement. In the process of *"Notice and report presence and absence"* two identified goals can be identified which are *"Comply with legislation"* and *"Prevent student dropouts"* and two identified goals of measurement which are "*Measure the effectiveness of reporting"* and "*Measure the effectiveness of mentor support*".

Next to the goals the process also contains requirements for achieving these goals. The requirements are the business rules that give insight in what actions are allowed or not allowed and what actions are mandatory.

Comply with legislation is one of the two goals of the process. The requirements to support this goal are derived from legislation (Leerplichtwet, 1969) and are the following:

- A student must be marked not present by a teacher if a student is not in class
- A student must be marked absent by an absentee worker after more than 14 days not present in class
- A student must be reported by an absentee worker within 5 days after being marked absent
- A student must be informed by an absentee worker when reported
- The parents of a student must be informed by an absentee worker if a student is younger than age 18
- A student must be marked as dropout after >60 days absent.

Prevent student dropouts is the second goal of the process. The following requirements are identified that support the second goal:

- A student must be assigned to a mentor by an administration worker
- A mentor must give support to a student who is absent
    - Giving support means seeking contact with the student, making agreements and motivate the student to return.

The goals of the process, goals of measurement, and the requirements are modeled. The result is a goal model shown in figure 6. The goals and goals of measurement are shown in green, and the requirements in the outer white boxes.

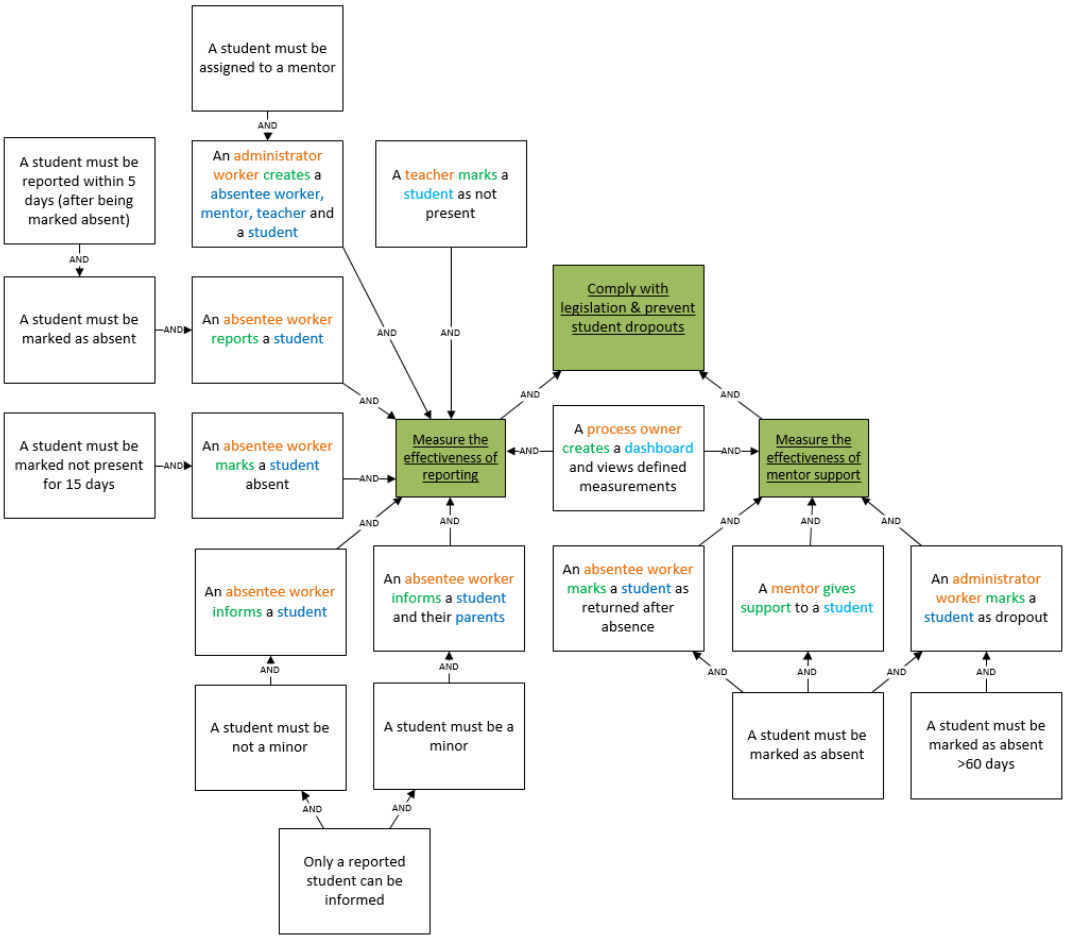


Figure 6: Goal model of the process: "Notice and report presence and absence"

The KPIs that need to be designed must relate to the goals of the process and the goals of measurement because this contributes to the consistency between them. Looking at the goals in the goal model, the following KPIs (in table 9) can be designed:

Table 9: KPIs related to goals and goals of measurement

KPI	Relates to the goal of process	Goals of measurement
Students reported in time	Comply with legislation	Measure the effectiveness of reporting
Absent students with mentor support	Prevent student dropouts	Measure the effectiveness of mentor support
Students returned after mentor support	Prevent student dropouts	Measure the effectiveness of mentor support
Students dropped out after mentor support	Prevent student dropouts	Measure the effectiveness of mentor support

In the next section the goal model is used as input to design an executable protocol model consisting of the business concepts and their various possible states.

8.2.2 Protocol model

Based on the goal model (with requirements) a protocol model of the case "*Notice and report presence and absence"* is developed.

In this protocol model the following protocol machines based on the business objects can be distinguished: *Student*, *Mentor*, *Teacher*, *Absentee worker*, and *Dashboard*. *Student* is the core protocol machine of the protocol model. The life cycle of the business object *Student* is shown in figure 7. The other protocol machines are described in the following sections and the complete protocol model can be found in appendix A.

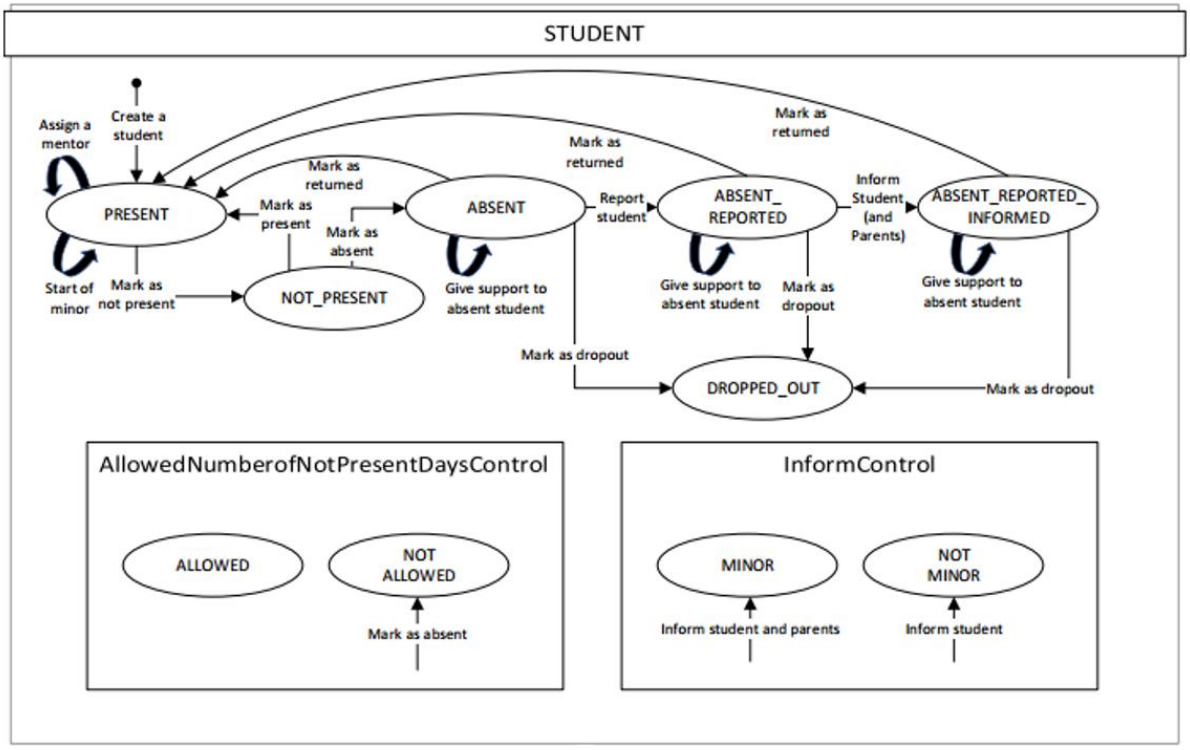


Figure 7: Protocol machine of concept Student

8.2.2.1 Protocol machine of concept Student

The protocol machine *Student* is the core object of the protocol model. A student can be in state *Present*, *Not_present, Absent, Absent_Reported, Absent_Reported_Informed*, or *Dropped_out*. These states have been identified from the case description. This protocol machine can handle the following events:

Table 10: Protocol machine of concept Student: events and actors responsible

Event	Actor responsible	Event	Actor responsible
Create a student	Administration worker	Give support to absent student	Mentor
Assign a mentor	Administration worker	Report student	Absentee worker
Start of minor	Administration worker	Inform student	Absentee worker
Mark as not present	Teacher	Inform student and parents	Absentee worker
Mark as present	Teacher	Mark as returned	Absentee worker
Mark as absent	Absentee worker	Mark as dropout	Administration worker

An actor is responsible for interaction with the protocol machine via the events associated with the actor, see table 10. A protocol machine consists of a business concept and events. In the code an OBJECT revers to a business concept. In this protocol machine this is *Student*.

Part of the event code (figure 8) and part of the protocol machine code (figure 9) of the business concept *Student* is presented below (the complete code can be found in appendix B):

```
EVENT Create a student
        ATTRIBUTES      Student: Student,
                        Student Name: String,
                        Start of minor: Date,
                        Student birthday: Date,
                        Mentor: Mentor
EVENT Assign a mentor
        ATTRIBUTES      Mentor: Mentor,
                        Student: Student
----------CODE CUT OFF, COMPLETE CODE CAN BE FOUND IN APPENDIX B----------
```

Figure 8: Part of the event code of the protocol machine for the business concept Student

```
OBJECT                  Student
        NAME            Student Name
        INCLUDES        StudentAttended, StudentInformed
        ATTRIBUTES      Student Name: String,
                        Start of minor: Date,
                        Student birthday: Date,
                        Mentor: Mentor,
                        Teacher: Teacher,
                        Absentee worker: AbsenteeWorker,
                        Mentor absence support: Boolean,
                        Student returned after absence: Boolean,
                        !Student age: Integer,
                        !Not present days: Integer,
                        !Absent days: Integer,
                        !Reported within days: Integer,
                        !Current state: String,
                        (Reported since date: Date)
        STATES          Present,
                        Not_present,
                        Absent,
                        Absent_Reported,
                        Absent_Reported_Informed,
                        Dropped_out
        TRANSITIONS     @new*Create a student=Present,
                        Present*Assign a mentor=Present,
                        Present*Mark as not present=Not_present,
                        Present*Start of minor=Present,
----------CODE CUT OFF, COMPLETE CODE CAN BE FOUND IN APPENDIX B----------
```

Figure 9: Part of the object code of the protocol machine for the business concept Student

When the protocol model is executed, the EVENT *Create a student* is presented to the environment. Executing this event allows creating a new instance of the object *Student*. The instance then gets state *@new*Create a student=Present.* Before allowing the model to accept the event attributes must be given. These are shown in figure 9 after ATTRIBUTES. For example, *Student Name* which must be a string value or *Start of minor* which must be a date value. Attributes with a "!" sign are automatically calculated via so called java callbacks and do not need to be filled in before executing event *Create a student*. Protocol machines have a lifecycle through different states, see figure 7. The protocol machines run back and forth between different states depending on the interaction executed via events. TRANSISTIONS at the bottom of figure 9 show the allowed transitions between states. For example, the model accepts a transition from state *Present* to *Not_present* and back to *Present*. Every change in states (transition) is initiated via an event and allows or not allows other events. Every executed event on the model, given attributes and (!) calculated attributes and state transitions are saved to a local data store. The INCLUDES relation means that for every instance of *Student* the instances of the dependent protocol machines *StudentAttended* and *StudentInformed* are created. INCLUDES allows for more flexibilization designing the model because it can be easily included in other protocol machines as desired.

Special attention to two controls that extend the behavior of protocol machine *Student*. In this protocol machine two controls are designed, these are *AllowedNumberofNotPresentDaysControl* and *InformControl*. They play a key role supporting the identified requirements, chapter 8.2.1. The first control has two states: *Allowed, Not Allowed*. *Allowed* means that a student is not present in class for less than 15 days and no events are available. *Not Allowed* means that a student is not present in class for more than 14 days and then event *Mark as absent* becomes available. This control supports the requirement of the goal model that a student must be marked absent after more than 14 days not present in class. The second control has also two states: *Minor, Not Minor.* If a student is a minor and thus <18 years of age then event *Inform student and parents* becomes available. If a student is >=18 years of age then event *Inform student* becomes available. This control supports the requirement that the parents of a minor student must be informed.

The java code designed for the two described controls are presented below in figure 10 and 11:

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class AllowedNumberofNotPresentDaysControl extends Behaviour {

public String getState() {
          if (this.getInteger("Not present days") >= 15) return "Not_allowed";
          else return "Allowed";
          }
}
```

Figure 10: Java code of the control AllowedNumberofNotPresentDaysControl

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class InformControl extends Behaviour {

public String getState() {
          if (this.getInteger("Student age") < 18) return "Minor";
         else return "Not_minor";
         }
}
```

Figure 11: Java code of the control InformControl

8.2.2.2 Protocol machine of concept Mentor

A protocol machine of the concept *Mentor* is designed. This protocol machine can be only in state *Created.* This protocol machine can handle the following events:

Table 11: Protocol machine of concept Mentor: events and actors responsible

Event	Actor responsible
Create a mentor	Administration worker
Assign a mentor	Administration worker
Create a student	Administration worker
Give support to absence student	Mentor

The complete event code (figure 12) and the protocol machine code (figure 13) for the business concept *Mentor* is presented below:

```
EVENT Create a mentor
        ATTRIBUTES      Mentor: Mentor,
                        Mentor Name: String

EVENT Assign a mentor
        ATTRIBUTES      Mentor: Mentor,
                        Student: Student

EVENT Create a student
        ATTRIBUTES      Student: Student,
                        Student Name: String,
                        Start of minor: Date,
                        Student birthday: Date,
                        Mentor: Mentor

EVENT Give support to absence student
        ATTRIBUTES      Mentor: Mentor,
                        Student: Student,
                        Mentor absence support: Boolean
```

Figure 12: Complete event code of the protocol machine for the business concept Mentor

```
OBJECT                  Mentor
        NAME            Mentor Name
        ATTRIBUTES      Mentor Name: String,
                        (Student: Student)
        STATES          Created
        TRANSITIONS     @new*Create a mentor=Created,
                        Created*Assign a mentor=Created,
                        Created*Create a student=Created,
                        Created*Give support to absence student=Created
```

Figure 13: Complete object code of the protocol machine for the business concept Mentor

The rest of the concepts are designed in the same way. To shorten the description of the model the rest of the concepts will be presented without the code with an exception of the concept Dashboard as this is a special concept because it is used for calculation. The code of all concepts can be found in appendix B.

8.2.2.3 Protocol machine of concepts Teacher, and Absentee worker

A protocol machine of the concept *Teacher,* and *Absentee worker* is designed. These protocol machines can be only in state *Created.* These protocol machines can handle the following events:

Table 12: Protocol machine concept Teacher events and actors responsible

Event	Actor responsible
Create a teacher	Administration worker

Table 13: Protocol machine concept Absentee worker events and actors responsible

Event	Actor responsible
Create an absentee worker	Administration worker

8.2.2.4 Protocol machine of concept Dashboard

A protocol machine of the concept *Dashboard* is designed. This is a special concept because it will be used to calculate designed KPIs and indicators and show all the values. The designed KPIs and corresponding java code will be described later. The KPIs calculated on this dashboard will be used for validation which means checking their 'well-defined' properties.

The protocol machine *Dashboard* can be only in state *Created.* This protocol machine can handle the following events:

Table 14: Protocol machine concept Dashboard events and actors responsible

Event	Actor responsible
Create dashboard	Process owner
Change reporting quarter	Process owner

The complete event code (figure 14) and the protocol machine code (figure 15) for the business concept *Dashboard* is presented below:

```
EVENT Create dashboard
        ATTRIBUTES      Dashboard: Dashboard,
                        Start of reporting quarter: Date,
                        !Dashboard Name: String
```

Figure 14: Complete event code of the protocol machine for the business concept Dashboard

```
OBJECT          Dashboard
    NAME        Dashboard Name
    ATTRIBUTES  Dashboard Name: String,
                Start of reporting quarter: Date,
                !Total number of students: Integer,
                !Sum of absent students: Integer,
                !Percentage absent students: Integer,
                !Sum of absent days of absent students: Integer,
                !Average number of absent days of absent students: Integer,
                !Sum of students need to be reported: Integer,
                !Sum of reported students: Integer,
                !Sum of students need to be informed: Integer,
                !Percentage students reported in time: Integer,
                !Percentage absent students with mentor support: Integer,
                !Percentage students returned after mentor support: Integer,
                !Percentage students dropped out after mentor support: Integer
```

Figure 15: Complete object code of the protocol machine for the business concept Dashboard

Special attention to the *Start of reporting quarter* attribute. During creation of the dashboard this attribute needs to be filled in and will set the reporting quarter. For example, filling in *1 SEP 17* will set the reporting quarter from *1 SEP 17* to *30 NOV 17*. This attribute of the dashboard corresponds to the TIME concept in the designed conceptual model in the literature phase of this study (figure 3). This allows for comparing KPI values over different periods of time.

8.2.2.5 Responsible actors

Actors are responsible to interact with the protocol machines. In every business concept described above a table with events and the responsible actor is presented.

From the actor point of view a business concept is called a behavior. The actor can interact with behaviors and via the events associated with the actor.

A description of all the actors, behaviors, and events is given per business concept description. But to get a clearer overview the corresponding code is presented below in figure 16.

```
ACTOR Absentee worker
        BEHAVIOURS      Student
        EVENTS          Mark as absent,
                        Report student,
                        Inform student,
                        Inform student and parents,
                        Mark as returned

ACTOR Administration worker
        BEHAVIOURS      Student, Mentor, Teacher, AbsenteeWorker
        EVENTS          Create a student,
                        Create a mentor,
                        Create a teacher,
                        Create an absentee worker,
                        Start of minor,
                        Assign a mentor,
                        Mark as dropout

ACTOR Teacher
        BEHAVIOURS      Student
        EVENTS          Mark as present,
                        Mark as not present

ACTOR Mentor
        BEHAVIOURS      Mentor, Student
        EVENTS          Give support to absence student

ACTOR Process owner
        BEHAVIOURS      Dashboard
        EVENTS          Create dashboard,
                        Change reporting quarter
```

Figure 16: Complete actor code for all protocol machines

8.3 Modeling semantics to design KPIs and validate their properties

ExtREME comprises modeling semantics that support the design of KPIs and validation of their properties. The following sections describe these steps. These steps involve goal modeling (with requirements) and refinement, simulation of the designed protocol model and validation of KPIs and their properties. The described steps below are performed in the case study.

8.3.1 Refinements of goals and requirements modeling

This step concerns identifying the goals of the process. The result of this step is described in chapter 8.2.1. This is the goal model with requirements, see figure 6. A document of an embedded business process formed the basis for goal modeling. The document is iteratively analyzed in search of (refinements of) goals and requirements of the process. The identified goals and requirements were then discussed with the process owner. The process owner is the source of the used document (Verzuimprotocol ROCvA). This person has well knowledge of the process and requirements (business rules).

8.3.2 Simulation of the protocol model via traces

The next step involves executing the protocol model. The designed protocol model (a part of which is shown in figure 7) is executed with artificial data. This is called simulation of the model. The data consists of instances of business concepts (objects) as protocol machines.

The data is created via the actors and allows to interact with the protocol machines. The goal is to create all kinds of different instances of objects in different states with different values of attributes.

These instances can be used for validation of KPI properties by counting the different objects, states, and values of attributes.

A part of the created artificial data is presented in figure 17 (the complete data can be found in appendix C):

```
INSTANCE : AbsenteeWorker_Richard = 1
        BEHAVIOUR : AbsenteeWorker = Created
                AbsenteeWorker Name : String = AbsenteeWorker_Richard

INSTANCE : KPIs Absentee Process = 3
        BEHAVIOUR : Dashboard = Created
                Dashboard Name : String = KPIs Absentee Process
                Start of reporting quarter : Date = 1 Aug 2017

INSTANCE : Mentor_Sjaak = 4
        BEHAVIOUR : Mentor = Created
                Mentor Name : String = Mentor_Sjaak
                Student : Student = 17

INSTANCE : Student_Daan = 21
        BEHAVIOUR : Student = Present
                Student Name : String = Student_Daan
                Start of minor : Date = 1 Sep 2017
                Student birthday : Date = 1 Jan 2000
                Mentor : Mentor = 6
                Teacher : Teacher = 22
                Absentee worker : AbsenteeWorker = 1
                Mentor absence support : Boolean = true
                Student returned after absence : Boolean = true
                Reported since date : Date = 15 Nov 2017
        BEHAVIOUR : StudentAttended = @new
                Not present since date : Date = 31 Oct 2017
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
                Informed date : Date = 15 Nov 2017
        BEHAVIOUR : InformControl = Minor
----------CODE CUT OFF, COMPLETE CODE CAN BE FOUND IN APPENDIX C----------
```

Figure 17: Part of the artificial data used for simulation

To simulate the model and to create artificial data or work with data already in the local data store traces are designed. The traces are used during simulation of the model. These traces help to understand how to go through the "*Notice and report presence and absence"* process. See figure 18.

1. The "Administration worker" creates an "AbsenteeWorker"
2. The "Administration worker" creates a "Mentor"
3. The "Administrator worker" creates a "Teacher"
4. The "Administrator worker" creates a "Student" either via the object "Mentor" or object "Student"
   - 4.1. The "Administrator worker" sets the "Student Name", "Start of minor" date (*which is used for quarterly reporting*), "Student birthday" date (*which is used to calculate age and determine if a "Student" is a minor*) and assigns an existing "Mentor"
   - 4.2. The "Administrator worker" can create additional "Mentors" and can assign existing "Students" to other existing "Mentors"
   - 4.3. In "Student" state "Absent", "Absent_Reported", or "Absent_Reported_Informed" if a "Student" drops out (>60 days absent) the "Administrator worker" must mark a "Student" as "Dropped_out"
5. The "Teacher" marks a "Student" as "Not_present" and the "Not present days"-counter is started (*for testing purposes entering a "Not present since date" of 15 days ago is allowed*)
   - 5.1. In the "Student" state "Not_present" the "Teacher" can mark a "Student" as "Present"
   - 5.2. If the "Not present days"-counter reaches "15" days the "Absentee worker" must mark the "Student" as "Absent" *(15 days limit is determined by law).* The "Student" then gets state "Absent"
   - 5.3. In the "Student" state "Absent", "Absent_Reported" or "Absent_Reported_Informed" if "Student" returns from absence the "Absentee worker" must mark the "Student" as returned and set the boolean returned after absence to "True" (*for testing purposes "True" is not forced as default)*
6. The "Mentor" must give support to an absent student" if a "Student" is in state "Absent", "Absent_Reported" or "Absent_Informed" either via the object "Mentor" or object "Student" (the "Mentor" sets the boolean "Give support to absent student" to "true" (*for testing purposes "True" is not forced as default)*
7. The "Absentee worker" reports a "Student" in state "Absent" and "Student" then gets state "Absent_Reported"

7.1. The “Absentee worker” must report a “Student” within 5 days as soon as the “Student” is in state “Absent” *(5 days limit is determined by law)*

8. If a “Student” is in state “Absent_Reported” the "Absentee worker" must "Inform student" if "Student" (=>18 year of age) and must “Inform student and parents” if "Student" (<18 year of age). “Student” gets state “Absent_Reported_Informed”
9. If a “Student” returns from absence the “Absentee worker” marks a “Student” in state "Absent", "Absent_Reported" or "Absent_Reported_Informed" as returned by setting the boolean “Student returned after absence” to “True” (*for testing purposes "true" is not forced as default*)
10. The “Process owner” creates a “Dashboard” to view PI and KPIs about the absentee process.

Figure 18: Traces for the “Notice and report presence and absence” process

The use of the traces result in a local datastore consisting of (changed) artificial data. The data forms input for the designed KPIs because without data the KPIs cannot be validated. Another result is that the data can be used by others to replicate simulation of the model.

9. Validating the properties facing the goals of the process and the indicators

Two related artefacts are designed, an executable protocol model and the corresponding goal model for validation of KPIs *“Notice and report presence and absence”*. These models can demonstrate to the process owners if the KPIs poses the ‘well-defined’ properties and if they are sufficiently *reliable* and *improvement oriented*.

Formal definitions of KPIs are identified during the literature phase of this study, see chapter 4 table 7. In the model KPIs are designed but it is unclear if they are ‘well-defined’ in respect to their properties: *improvement-oriented, linear, quantifiable, reliable, and sensitive.* It is needed to validate the properties. Validation is the next step used in the case study. During validation the conceptual model (figure 3) from the literature phase is used as a reference to determine the validity of properties.

The designed dashboard protocol machine contains the designed indicators and KPIs (chapter 8.2.2.4). The indicators and KPIs are designed and calculated via java code which will be presented in the next sections. The designed indicators and KPIs are shown in figure 19.

Dashboard Name	KPIs Absentee Process
Start of reporting quarter	1 Aug 2017
Total number of students	15
Sum of absent students	6
Percentage absent students	40
Sum of absent days of absent students	148
Average number of absent days of absent students	24
Sum of students need to be reported	3
Sum of reported students	6
Sum of students need to be informed	6
Percentage students reported in time	44
Percentage absent students with mentor support	66
Percentage students returned after mentor support	20
Percentage students dropped out after mentor support	6

Figure 19: Designed indicators and KPIs

In figure 19 “Dashboard Name” is the name of the dashboard which automatically gets the default value “KPIs Absentee Process”. The “Start of reporting quarter” in this case is “1 Aug 2017”.

There is no “End of reporting quarter” because it is always a time frame of three months. Thus, in this case all the values shown are calculated between “1 Aug 2017” and “31 Oct 2017”.

It is possible to change the reporting quarter. If this is done only the values for that period will be presented. This allows for comparing the values between different time periods. For example, the value of the current reporting quarter of *Percentage absent students returned after mentor support* is 20%. If a different reporting quarter shows a value of 25% than the mentor support for that specific quarter is ‘better’.

A business process can have many indicators, but the focus lies on KPIs that correspond to a business goal. This is because in the conceptual model created in the literature phase is concluded that an indicator must correspond to a goal in order to be identified as a KPI. See the relation in figure 4 between the concept KPI and business goal. So, the question is which of the indicators in figure 19 correspond to the goals of the process. The goals of the process and goals of measurements are described in chapter 8.2.1.

With the knowledge in the conceptual model in figure 4 and looking at the designed indicators in figure 19 it is concluded that the following indicators can be identified as KPIs because they correspond to the goals of the process. The others are only indicators as they do not correspond to a goal of the process:

- *Percentage students reported in time*
- *Percentage absent students with mentor support*
- *Percentage students returned after mentor support*
- *Percentage students dropped out after mentor support.*

This study focuses on the goal *preventing student dropouts* because an education institute cannot exist without students and therefore this goal is chosen above *comply with legislation*. This means that the following KPIs will be used for validation (of their properties):

- *Percentage absent students with mentor support*
- *Percentage students returned after mentor support*
- *Percentage students dropped out after mentor support.*

The three KPIs share the same goal of measurement which is *Measure the effectiveness of mentor support*. The rest in figure 19 are only indicators because they do not correspond to a goal of the process and therefore will not be used for validation as it is only possible to validate properties of KPIs and not indicators.

9.1 Percentage absent students with mentor support

This KPI counts all *Student* objects (business concepts) with state *Absent, Absent_Reported,* and *Absent_Reported_Informed* and attribute *Mentor absence support* with value *true.*

In figure 20 the corresponding java code is presented.

```
public int getPercentageAbsentStudentsWithMentorSupport() {

                int SumOfAbsentStudentsWithMentorSupport = 0;
                int TotalNumberOfAbsentStudents=getSumOfAbsentStudents();
                int PercentageAbsentStudentsWithMentorSupport = 0;

                Date StartRQ=GetStartQuarter();
                Date EndRQ=GetEndQuarter();

                Instance[] absentstudents = selectInState("Student", "Absent");
```

```
for (int i = 0; i < absentstudents.length; i++) {
Date MinorDate=absentstudents[i].getDate("Start of minor");

        if (absentstudents[i].getBoolean("Mentor absence support") &&
        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
        SumOfAbsentStudentsWithMentorSupport +=1; }
}

Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
for (int i = 0; i < absent_reportedstudents.length; i++) {
Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");

        if (absent_reportedstudents[i].getBoolean("Mentor absence support") &&
        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
        SumOfAbsentStudentsWithMentorSupport +=1; }
}

Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
for (int i = 0; i < absent_informedstudents.length; i++) {
Date MinorDate=absent_informedstudents[i].getDate("Start of minor");

        if (absent_informedstudents[i].getBoolean("Mentor absence support") &&
        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
        SumOfAbsentStudentsWithMentorSupport +=1; }
}

if (TotalNumberOfAbsentStudents == 0) {
PercentageAbsentStudentsWithMentorSupport = 0;
}
else
{
PercentageAbsentStudentsWithMentorSupport = ((100)*(SumOfAbsentStudentsWithMentorSupport) /
(TotalNumberOfAbsentStudents));
}
return PercentageAbsentStudentsWithMentorSupport;
}
```

Figure 20: Java code of KPI Percentage absent students with mentor support

For clarification reasons the properties from Roubtsova & Michell (2014) that we need to validate are presented again:

- A KPI should be in a *quantifiable* form
- A KPI needs to be *sensitive* to changes
- A KPI should be *linear*
- A KPI should be semantically *reliable*
- A KPI should be *oriented to improvement*, not to conformance to plans.

Quantifiable. This property is validated by the way the KPI is designed. This is because when executing the protocol model, it allows for selecting and counting the instances of Students in the model in specific states with a specific variable set to *true*, that is *Mentor absence support*. See the functions “selectInState("Student", "Absent");”, “selectInState("Student", "Absent_Reported");” and “selectInState("Student", "Absent_Reported_Informed");” in figure 20. This allows for selecting and counting the instances in three different states with *Mentor absence support* set to the value *true* and save the result in the variable *SumOfAbsentStudentsWithMentorSupport*. The select function enables modeling of quantifiable KPIs. Therefore, it is concluded that the property *Quantifiable* is validated by design for this KPI.

Sensitive. It is always needed to validate this property when a KPI is presented in rates/percentages because when the change in values is to ‘small’ it is not always showing a different result of the KPI value. This is for example the case when working with percentages with no decimals like this KPI or when counting and dividing shows the same result.

Counting objects and variables will result in a value of the KPI of for example 66 (which means 66%). This is calculated by counting all absent students with mentor support divided by counting the total number of absent students.

See the following function in figure 20: *((100)* (SumOfAbsentStudentsWithMentorSupport) / (TotalNumberOfAbsentStudents)).* The value of 66 is achieved when the values of the attributes are for example 4 / 6 = 66%. If the values would change to 2 / 3 the result would also by 66% and then to KPI could be interpreted as not being sensitive (because the value of 66 did not change). To validate sensitivity the values can be manually changed to 5 / 6 which will result in 83%. Therefore, it is concluded that the property *Sensitive* is validated for this KPI.

Linear. This property is validated by the way the KPI is designed. This is tested by executing the model with populated objects. In this case the number of absent students and the number of students with mentor support. These values are the arguments of the formula: *((100)* (SumOfAbsentStudentsWithMentorSupport) / (TotalNumberOfAbsentStudents)).* It is the formula that makes this KPI linear. Therefore, it is concluded that the property *Linear* is validated for this KPI.

Reliable. This property can be validated by looking at the assumptions that need to be made about a business process to derive the KPI value. A KPI is reliable if the set of additional assumptions about the business process is empty. This KPI is based on a well-documented and embedded business process that must support legislation requirements next to preventing student dropouts. Therefore, this process is imposed to exist within the organization and the output of the process is regularly audited by external entities like the government. Also, the documentation used (Verzuimprotocol ROCvA) is submitted by the process owner within the case organization. This person has well knowledge of the process and required business rules. The designed KPI is based on this well-documented information and there were no assumptions made during the design of this KPI. Therefore, it is concluded that the property *Reliable* is validated for this KPI.

Improvement oriented. This property can be validated by looking at the number of manipulative scenarios. A KPI is only oriented to improvement if the set of manipulative scenarios is empty. It is identified that there are three (potential) manipulative scenarios.

Manipulative scenario 1: An absentee worker does not mark a student absent after 15 days not present in class. Then the value of this KPI does not reflect the actual situation because the current number of absent students is (temporarily) incorrect until marked. Although this undesired scenario is ‘seen’ (the value is not ‘0’) and is presented in the indicator *Sum of students need to be reported,* the value of this KPI still does not reflect the actual number of absent students and therefore not the correct calculated percentage value. Only if the value of the indicator *Sum of students need to be reported* is ‘0’ then this KPI shows the correct value and reflects the actual situation.

Manipulative scenario 2: Another manipulative scenario is where a teacher does not mark a student not present. Then the *not present counter* of 15 days will not start and then the value of the KPI will not reflect the actual situation.

Manipulative scenario 3: This KPI counts Student objects in any of the three absent states (Absent, AbsentReported, AbsentReportedInformed). There is a business rule in the form of a requirement that states that a student must be marked as dropout if absent days > 60. This is not being enforced. So, this could mean that this KPI counts also the students that are absent > 60 days which is not correct because they should be marked as dropout and not being counted.

No other manipulative scenarios were found during simulation of the model.

In table 15 the result of the validation of the KPI *Percentage absent students with mentor support* is shown. By executing the protocol model others can replicate the results. To do this the complete code of the protocol model can be used and be found in appendix B.

Table 15: Validation of properties KPI "Percentage absent students with mentor support"

Property	Validated
Quantifiable	YES (by design)
Sensitive	YES
Linear	YES (by design)
Reliable	YES
Improvement oriented	NO

The other two KPIs are validated in the same way as the KPI *Percentage absent students with mentor support.* It is concluded that the property validation result in table 15 is identical for the other two KPIs. This is because these KPI's are also *Quantifiable* and *Linear* by the way they are designed. The property *Sensitivity* is also validated because their values are presented in percentages too. *Reliable* can also be validated because the other two KPIs use the same basis of information about the process as the first KPI. The *improvement oriented* cannot be validated because of the same manipulative scenarios 1 and 2. Scenario 3 does not apply here because no students in any of the three absent states are counted. See table 16 for a summary of the properties validation of all KPIs.

Table 16: Summary of properties validation of all KPIs

KPI	Quantifiable	Sensitive	Linear	Reliable	Improvement oriented
Percentage absent students with mentor support	YES	YES	YES	YES	NO
Percentage students returned after mentor support	YES	YES	YES	YES	NO
Percentage students dropped out after mentor support	YES	YES	YES	YES	NO

It is concluded that all the properties of the three KPIs can be validated except *improvement oriented* because of the identified and described (manipulative) scenarios.

10. Conclusion empirical phase

The empirical phase consisted of a case study and was directed to find if the conceptual model (from the literature phase) and the ExtREME-method allows to design KPIs, validate 'well-defined' properties and identification of scenarios where some properties of KPIs are not met. The empirical phase also aimed to answer the sub research question: *Do the KPIs designed for the case possess the properties of a 'well-defined' KPI?* (SRQ5).

The conceptual model consisting of seven concepts and five properties designed in the literature phase helped executing the empirical phase of this study because it gave insight in what information in the case study was needed to be able to design and validate (properties) of KPIs. Information about (among others) a business process, business goals, and business objects with attributes.

In table 17 we summarize all the concepts from the conceptual model and how they were used in the case study.

Table 17: Conceptual model usage in the case study

Conceptual model concepts	Case study usage
Business process	Process: "*Notice and report presence and absence"*
Business goal	Prevent student dropouts
Business strategy	Designed KPIs measuring degree of achievement of strategy (translated in goal)
Business objects	(Executive) protocol model with (instances) of objects i.e. Students
Attributes	Attributes of objects like *Name* or *Start of minor* of i.e. a Student
Time	The Dashboard object in the protocol model that allows quarterly reporting
KPI	The three designed KPIs supporting the goal prevent student dropouts

The conceptual model can be reused in future case studies because the concepts remain the same. It is only the case usage of the model that changes the values of the concepts but not the concepts itself.

Do the KPIs designed for the case possess the properties of a 'well-defined' KPI?

The case proves that the ExtREME-method enabled the design of an executable protocol model with several KPIs and indicators. The case study focused on the *prevent student dropouts* goal of the process *notice and report presence and absence*. Of all designed indicators three KPIs were identified because they corresponded to the focused goal. The method enabled the validation of the 'well-defined' properties of these KPIs and it is concluded that the three designed KPIs possess four of the five properties of a 'well-defined' KPI. This means that the properties *Quantifiable, Sensitive, Linear,* and *Reliable* could be validated but the *Improvement oriented* property could not. This is because three manipulative scenarios were found that undermine this property.

Special attention to the reliability property because of the following. During the case study no norms about mentor support were found. This means that the value of a KPI cannot be compared against a norm but only to other reporting period values. However, how well the mentors are performing is still possible to determine for example when in a specific reporting quarter X, the value of students that dropout after mentor support is 6% and in another quarter Y it is 15%. It can then be easily determined that the mentor's performance was better in quarter X. However, the use of norms (derived from business strategy) would contribute to the level of reliability.

SRQ6 focuses on the validation of the method of KPI design and will be answered in the following *Discussion and reflection* section.

11. Discussion and reflection

The goal of this section is to discuss the study results and reflect on the quality of the followed process during this study and on the quality of the conclusions. Together this forms the answer to the last research question SRQ6: *How to evaluate the method of KPI design with design science?*

The aim of this research was to find ways to design and validate 'well-defined' KPIs corresponding with a current challenge in an education institute. The ExtREME-method was chosen because of its modeling semantics like goal modeling and protocol modeling. During this study a replication of the ExtREME-method was performed for a specific case study in an education institute in the Netherlands.

The last research question entails evaluation with design science. Design science focuses on extending the boundaries of human and organizational capabilities by creating new and innovative artifacts (Hevner et. al., 2004).

ExtREME can be named as an instantiation of design science method for the design of business processes and their KPIs and should be evaluated. This does not mean the evaluation of the ExtREME-method itself but rather the way the models were created using this method. In other words, an evaluation of case study design results.

Hevner et. al (2004) created a conceptual framework with seven guidelines and contend that each of these guidelines should be addressed in some manner for design-science research to be complete. The use of this framework is chosen because it helps to evaluate and reflect on the case study design results. This will be done by looking at the research "completeness" by reviewing and giving a description on how this study addresses each of the seven guidelines for the case study design results.

The following descriptions for each of the seven guidelines together form the evaluation and answer to SRQ6. These form also the input for reflection on the quality of the followed process during this study and on the quality of the conclusions. This will be described after the guidelines.

Design as an artifact (guideline 1): *Design-research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.*
This guideline is addressed because the used method allowed designing an artifact in the form of an executable model of KPIs on a relevant model of a business process.

Problem relevance (guideline 2): *One of the objectives of design-research is to develop technology-based solutions to important and relevant business problems.*
KPIs address an important organization problem of measurement and control of a business process and therefore it is concluded that this guideline is addressed.

Design evaluation (guideline 3): *The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.*
During the literature phase of the study the definitions of properties that form a 'well-defined' KPI were thoroughly collected and put in a conceptual model together with the concepts that form a KPI. The conceptual model helped identifying which information was needed in the case study and formed the basis for the design and validation of KPIs.

An internal business document that comprehensively describes the business process was used to form a basis for the KPI design. A list of the requirements was identified from the document and used to create a goal model with requirements.

An executive protocol model that can be executed via the tool ModelScope was developed based on the goal model and filled with artificial data. The use of artificial data in the model allows others to replicate findings and enables demonstration of each requirement for the business process and for the measurement.

This allowed for validation of the properties via simulation by execution of the model. During execution special attention was payed to the *reliability* and *improvement orientated* (properties) of KPIs because these can (potentially) be influenced by the behavior of people.

The application of a replication of the ExtREME-method as a case study led to a solution of the design problem. The design problem comprises the design of 'well-defined' KPIs for an education institute in the Netherlands. It gave insight in which KPIs can be designed for a specific educational business process and how they relate to the 'well-defined' properties. It gave insight in that all the properties of the designed KPIs can be validated except the improvement oriented property and why (manipulative scenarios).

Based on the above design evaluation it is concluded that the designed artifact (model) comprises the utility, quality and efficacy and that this guideline is addressed.

Research contribution (guideline 4): *Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.*
The design artifact in the form of an executable protocol model with KPIs is a contribution to research. This is because it shows a solution of an unclear problem. Namely, how to assess properties of KPIs in an education institute. Another contribution is a replication of a case of KPI design using an existing method. It is therefore concluded that this guideline is addressed.

Research rigor (guideline 5): *Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.*
The following statements can be made concerning research rigor during construction of the designed artifact. The protocol model is deterministic. Determinism is achieved by communicating sequential processes (CSP) composition of protocol machines by event synchronization. This means that protocol machines only accept specific events if they are in a specific state and that during executing of the model the result will always be the same. For example, when others try to replicate the findings and try to test and reproduce the protocol model the result will be the same. The designed traces also contribute to enabling others to replicate the findings.
The following statements can be made about evaluation of the designed artifact. The properties of a 'well-defined' KPI are previously identified. So, during execution of the protocol model it is known what to test. There is also a data collection database in the form of a txt-file that can be used for calculations of the KPIs used in the case study. The protocol model also contains the formulas used for calculation. The formulas and the data collection database can be found in respectively appendix B (see dashboard.java in section callbacks) and C. This all contributes to the evaluation (by others) of the designed artifact. It is therefore concluded that this guideline is addressed.

Design as a search process (guideline 6): *The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.*
The search process during this study was driven by the needs of KPI design. Available means of search were several contact moments with a process owner to collect information about a business process that could be used for KPI design and validation. This resulted in obtaining a document that well described the process and formed the basis of input for the case study.

Based on the document the goal model was designed without process requirements to reduce the complexity of the first design iteration. Subsequently the first version of the executable model was built without KPIs. Then a list of requirements was identified (based on the document) and the goal- and protocol model were altered with requirements and KPI-calculations. Next the traces were designed (second iteration). The model was then discussed with the process owner and based on the discussion two concepts (Teacher and Absentee worker) were added to the model (third iteration). This was because concepts have a responsibility in the business process but this responsibility was not (yet) modeled. Then the search was stopped for validation. After the first validation several iterations were made in altering KPI formulas because they calculated the wrong business objects (and attributes) and gave a distorted view of KPI values. This continued until the KPIs showed the right values. This was checked by manually counting objects and performing calculations and compare it with the KPI results. Then a validation moment revealed three manipulative scenarios. The search continued with another contact moment with the process owner to discuss the found manipulative scenarios. The traces were used to present the process owner where the process (and KPIs) could be manipulated. The process owner confirmed the manipulative scenarios and no other manipulative scenarios were identified by the process owner. After that the search process was stopped and no further alterations were made to the model. Based on the above description it is concluded that this guideline is addressed.

Communication of research (guideline 7): *Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.*
During the design as a search process the model was presented to a process owner. The process owner does not belong to a technology-oriented- or a management-oriented audience, but this person has the responsibility for the performance of the process in realizing its objectives measured by the KPIs. This person also has the authority and ability to make changes.

The artifact is not presented to a broad technological- and management-oriented audience, but the artifact is ready for presentation to both audiences.

The research can be presented in the form of a presentation beginning with explaining the process and requirements first and then presenting the goal model. Then a demonstration of the protocol model and KPIs and why some properties are or are not met.

During the presentation of the model at least the following questions should be asked to the audience:

Table 18: Questions to ask to the audience

Question	Audience
Does the model clarify what it enables and where it can be used for?	Both technological- management oriented
Is the organizational context of usage of the model clear?	Both technological- management oriented
Is the presented model giving enough insight in what resources should be committed for constructing and using the model?	Management oriented
Is the process that enabled the construction and evaluation of the model clear?	Technological oriented
Is the presented model giving enough insight in how to implement and use it?	Technological oriented
Are there ways to prevent the identified manipulative scenarios?	Both technological- management oriented
Are there other manipulative scenarios that not have been identified?	Both technological- management oriented

The quality of a study and their conclusions are key factors to address during a study. This also applies to what is added to the current body of knowledge of a research field.

Based on the descriptions of guideline 3, 5, and 6 it is concluded that the quality of this study and the followed process are assured as much as possible and this implicates that the substantiated conclusions are of a certain degree of quality. Looking at guideline 1, 2, and 4 it is concluded that this study contributes to the existing body of knowledge.

This study has however also some limitations. Guideline 7 is not addressed because the model is not presented to a broad technological- and management-oriented audience. If this was done the obtained feedback could be processed and applied to make the model better. Although presenting to a broad audience was not the main aim of this study, obtaining and processing feedback from it could have made the model more practically implementable and more valuable for the case organization. A positive point to mention however, is that the model is ready for presentation to both audiences. Also, seven questions for both audiences are proposed that help in obtaining feedback on the model.

A second limitation is that the study is designed around a case study. Because of this the generalizability is limited, the so called external validity. The results shown in this study are valid for the specific case organization but not necessarily for other education institutes in the Netherlands because the business process can be slightly different. However, the way the results are achieved are valuable for other institutes because they allow them to get insight into how the results were achieved. This specifically applies for the used modeling semantics because others can use it during their own case study. The created models form examples to others and the designed conceptual model is generic which means that others can use it as a starting point for obtaining the needed information for their own case study.

12. References

Chandler, A. (1962). Strategy and Structure: Chapters in the History of the American Industrial Enterprise. MIT.

Doorewaard, H., & Verschuren, P. (2016). *Het ontwerpen van een onderzoek.* Amsterdam: Boom uitgevers Amsterdam.

Fortuin, L. (1988). Performance indicators - Why where and how? *European Journal of Operational Research*, 1-9.

Hevner, A., March, S., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS quarterly 28 (1)*, 75-105.

Kueng, P. (2000). Process performance measurement system: a tool to support process-based organizations.

Leerplichtwet 1969. (1968). Retrieved from Overheid: http://wetten.overheid.nl/BWBR0002628/2017-07-01

Lohman, C., Fortuin, L., & Wouters, M. (2004). Designing a performance measurement system: A case study.

Maté, A., Trujillo, J., & Mylopoulos, J. (2017). Specification and derivation of key performance indicators for business analytics: A semantic approach. *Elsevier Data & Knowledge Engineering*.

Morris, T., & Wood, S. (1991). Testing the survey method: continuity and change in British industrial relations. *Work Employment and Society*, 259-82.

Mylopoulos, J. (2009). Conceptual modeling and Telos.

Neely, A., & Platts, K. (2005). Performance measurement system design: A literature review and research agenda.

Neely, A., Richards, H., Mills, J., Platts, K., & Bourne, M. (1997). Designing performance measures: a structured approach.

Popova, V., & Sharpanskykh, A. (2010). Modeling Organizational Performance Indicators.

Porter, M. (1980). Competitive Strategy.

Roubtsova, E. (2016). *Interactive Modeling and Simulation in Business System Design.* Cham: Springer.

Roubtsova, E., & Michell, V. (2014). KPIs and Their Properties Defined with the EXTREME Method.

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering.

SAMBO-ICT. (2012). Retrieved from De Triple-A architectuur: https://triplea.sambo-ict.nl/wiki/index.php/De_Triple_A-architectuur

Scholten, C. F. (2017). *Specification, Modeling and Validation of KPIs and their required properties* (Master's thesis). Open Universiteit Nederland. Retrieved from `http://hdl.handle.net/1820/9332`

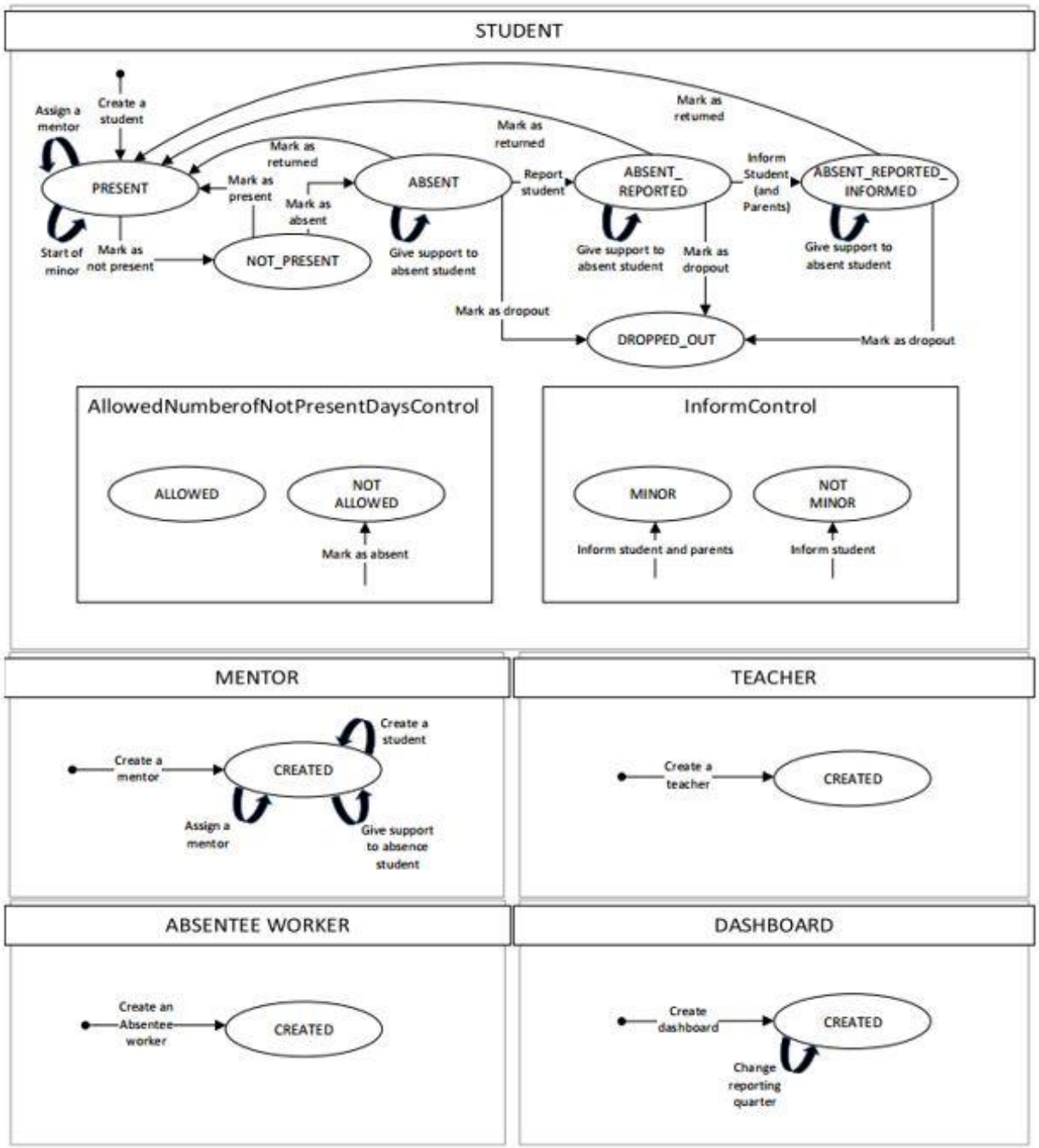
Sinclair, D., & Zairi, M. (1995). Effective process management through performance measurement, part III: An integrated model of total quality-based performance measurement. *Business Process Re-engineering & Management Journal*, 50-65.

Strecker, S., Frank, U., Heise, D., & Kattenstroth, H. (2012). MetricM: A modeling method in support of the reflective design and use of performance measurement systems.

W. Eckerson, W. (2009). Performance management strategies.

Yin, R. K. (2009). Case study research: Design and methods.

Appendix A: Protocol model



Appendix B: Protocol model code

```
MODEL ROCvA_Verzuimproces

# BEHAVIOUR definitions

BEHAVIOUR !AllowedNumberofNotPresentDaysControl
        TYPE ESSENTIAL
        STATES Allowed, Not_allowed
        TRANSITIONS Not_allowed*Mark as absent=@any

BEHAVIOUR !InformControl
        TYPE ESSENTIAL
        STATES Minor, Not_minor
        TRANSITIONS Minor*Inform student and parents=@any,
                    Not_minor*Inform student=@any

BEHAVIOUR       StudentAttended
        INCLUDES        AllowedNumberofNotPresentDaysControl
        ATTRIBUTES      (Not present since date: Date)

BEHAVIOUR       StudentInformed
        INCLUDES        InformControl
        ATTRIBUTES      (Informed date: Date)

# OBJECT definitions

OBJECT          Student
        NAME            Student Name
        INCLUDES        StudentAttended, StudentInformed
        ATTRIBUTES      Student Name: String,
                        Start of minor: Date,
                        Student birthday: Date,
                        Mentor: Mentor,
                        Teacher: Teacher,
                        Absentee worker: AbsenteeWorker,
                        Mentor absence support: Boolean,
                        Student returned after absence: Boolean,
                        !Student age: Integer,
                        !Not present days: Integer,
                        !Absent days: Integer,
                        !Reported within days: Integer,
                        !Current state: String,
                        (Reported since date: Date)
        STATES          Present,
                        Not_present,
                        Absent,
                        Absent_Reported,
                        Absent_Reported_Informed,
                        Dropped_out
        TRANSITIONS @new*Create a student=Present,
                        Present*Assign a mentor=Present,
                        Present*Mark as not present=Not_present,
                        Present*Start of minor=Present,
                        Not_present*Mark as present=Present,
                        Not_present*Mark as absent=Absent,
                        Absent*Give support to absence student=Absent,
                        Absent*Mark as dropout=Dropped_out,
                        Absent*Mark as returned=Present,
                        Absent*Report student=Absent_Reported,
                        Absent_Reported*Report=Absent_Reported,
                        Absent_Reported*Give support to absence student=Absent_Reported,
                        Absent_Reported*Mark as dropout=Dropped_out,
```

Absent_Reported*Mark as returned=Present,
Absent_Reported*Inform student=Absent_Reported_Informed,
Absent_Reported*Inform student and parents=Absent_Reported_Informed,
Absent_Reported_Informed*Inform=Absent_Reported_Informed,
Absent_Reported_Informed*Give support to absence student=Absent_Reported_Informed,
Absent_Reported_Informed*Mark as returned=Present,
Absent_Reported_Informed*Mark as dropout=Dropped_out

OBJECT Dashboard
NAME Dashboard Name
ATTRIBUTES Dashboard Name: String,
Start of reporting quarter: Date,
!Total number of students: Integer,
!Sum of absent students: Integer,
!Percentage absent students: Integer,
!Sum of absent days of absent students: Integer,
!Average number of absent days of absent students: Integer,
!Sum of students need to be reported: Integer,
!Sum of reported students: Integer,
!Sum of students need to be informed: Integer,
!Percentage students reported in time: Integer,
!Percentage absent students with mentor support: Integer,
!Percentage students returned after mentor support: Integer,
!Percentage students dropped out after mentor support: Integer

STATES Created
TRANSITIONS @new*Create dashboard=Created,
Created*Change reporting quarter=Created

OBJECT Mentor
NAME Mentor Name
ATTRIBUTES Mentor Name: String,
(Student: Student)
STATES Created
TRANSITIONS @new*Create a mentor=Created,
Created*Assign a mentor=Created,
Created*Create a student=Created,
Created*Give support to absence student=Created

OBJECT Teacher
NAME Teacher Name
ATTRIBUTES Teacher Name: String
STATES Created
TRANSITIONS @new*Create a teacher=Created,
Created*Mark as not present=Created,
Created*Mark as present=Created

OBJECT AbsenteeWorker
NAME AbsenteeWorker Name
ATTRIBUTES AbsenteeWorker Name: String
STATES Created
TRANSITIONS @new*Create an absentee worker=Created,
Created*Mark as absent=Created,
Created*Report student=Created,
Created*Report=Created,
Created*Mark as returned=Created,
Created*Inform student=Created,
Created*Inform student and parents=Created,
Created*Inform=Created

```
# EVENT definitions

EVENT Create a student
        ATTRIBUTES      Student: Student,
                        Student Name: String,
                        Start of minor: Date,
                        Student birthday: Date,
                        Mentor: Mentor

EVENT Create a mentor
        ATTRIBUTES      Mentor: Mentor,
                        Mentor Name: String

EVENT Create a teacher
        ATTRIBUTES      Teacher: Teacher,
                        Teacher Name: String

EVENT Create an absentee worker
        ATTRIBUTES      AbsenteeWorker: AbsenteeWorker,
                        AbsenteeWorker Name: String

EVENT Assign a mentor
        ATTRIBUTES      Mentor: Mentor,
                        Student: Student

EVENT Start of minor
        ATTRIBUTES      Student: Student,
                        Start of minor: Date

EVENT Mark as not present
        ATTRIBUTES      Student: Student,
                        Teacher: Teacher,
                        !Not present since date: Date

EVENT Mark as present
        ATTRIBUTES       Student: Student,
                         Teacher: Teacher

EVENT Mark as absent
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker

EVENT Mark as returned
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker,
                        Student returned after absence: Boolean

EVENT !Report student
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker,

EVENT Report
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker,
                        Reported since date: Date

EVENT !Inform student
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker

EVENT !Inform student and parents
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker
```

```
EVENT Inform
        ATTRIBUTES      Student: Student,
                        Absentee worker: AbsenteeWorker,
                        Informed date: Date

EVENT Give support to absence student
        ATTRIBUTES      Mentor: Mentor,
                        Student: Student,
                        Mentor absence support: Boolean

EVENT Mark as dropout
        ATTRIBUTES      Student: Student,

EVENT Create dashboard
        ATTRIBUTES      Dashboard: Dashboard,
                        Start of reporting quarter: Date,
                        !Dashboard Name: String

EVENT Change reporting quarter
        ATTRIBUTES      Dashboard: Dashboard,
                        Start of reporting quarter: Date

# ACTOR definitions

ACTOR Absentee worker
        BEHAVIOURS      Student
        EVENTS          Mark as absent,
                        Report student,
                        Inform student,
                        Inform student and parents,
                        Mark as returned

ACTOR Administration worker
        BEHAVIOURS      Student, Mentor, Teacher, AbsenteeWorker
        EVENTS          Create a student,
                        Create a mentor,
                        Create a teacher,
                        Create an absentee worker,
                        Start of minor,
                        Assign a mentor,
                        Mark as dropout

ACTOR Teacher
        BEHAVIOURS      Student
        EVENTS          Mark as present,
                        Mark as not present

ACTOR Mentor
        BEHAVIOURS      Mentor, Student
        EVENTS          Give support to absence student

ACTOR Process owner
        BEHAVIOURS      Dashboard
        EVENTS          Create dashboard,
                        Change reporting quarter
```

Callback functions

AllowedNumberofNotPresentDaysControl.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class AllowedNumberofNotPresentDaysControl extends Behaviour {

	public String getState() {
			if (this.getInteger("Not present days") >= 15) return "Not_allowed";
			else return "Allowed";
		}
}
```

CreateDashboard.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;

public class CreateDashboard extends Event {

	public void setDashboardName(EventValueAttribute attribute, Instance selected,
	  String subscript) {
		attribute.setString(DashboardInit.defaultDashboardName());
		attribute.setRule("WORD_CHAR_RULE", "Dashboard Name must be supplied");
	}

}
```

Dashboard.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class Dashboard extends Behaviour {

// Set the start and end reporting quarter date

	public Date GetStartQuarter() {

			Calendar cal=Calendar.getInstance();
			Date StartRQ = this.getDate("Start of reporting quarter");
			cal.setTime(StartRQ);
			return StartRQ;
	}

	public Date GetEndQuarter() {

			Calendar cal=Calendar.getInstance();
			Date StartRQ = this.getDate("Start of reporting quarter");
			cal.setTime(StartRQ);
			cal.add(Calendar.MONTH, 3);
			Date EndRQ = cal.getTime();
			return EndRQ;
	}
```

```
// Total number of students

	public int getTotalNumberOfStudents() {

		int Totalnumberofstudents = 0;

		Date StartRQ=GetStartQuarter();
		Date EndRQ=GetEndQuarter();

		Instance[] students = selectInState("Student", "@any");
		for (int i = 0; i < students.length; i++) {
			Date MinorDate=students[i].getDate("Start of minor");
			if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
			Totalnumberofstudents += 1; }
		}
		return Totalnumberofstudents;
	}

// Sum of absent students

	public int getSumOfAbsentStudents() {

		int Sumofabsentstudents = 0;

		Date StartRQ=GetStartQuarter();
		Date EndRQ=GetEndQuarter();

		Instance[] absentstudents = selectInState("Student", "Absent");
		for (int i = 0; i < absentstudents.length; i++) {
			Date MinorDate=absentstudents[i].getDate("Start of minor");
			if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
			Sumofabsentstudents += 1; }
		}

		Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
		for (int i = 0; i < absent_reportedstudents.length; i++) {
			Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");
			if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
			Sumofabsentstudents += 1; }
		}

		Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
		for (int i = 0; i < absent_informedstudents.length; i++) {
			Date MinorDate=absent_informedstudents[i].getDate("Start of minor");
			if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
			Sumofabsentstudents += 1; }
		}
		return Sumofabsentstudents;
	}

// Sum of reported students

	public int getSumOfReportedStudents() {

		int Sumofreportedstudents = 0;

		Date StartRQ=GetStartQuarter();
		Date EndRQ=GetEndQuarter();

		Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
		for (int i = 0; i < absent_reportedstudents.length; i++) {
			Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");
			if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
```

```
                        Sumofreportedstudents += 1; }
                }
                return Sumofreportedstudents;
        }

// Sum of absent days of all absent students

        public int getSumOfAbsentDaysOfAbsentStudents() {

                int Sumofabsentdaysofabsentstudents = 0;

                Date StartRQ=GetStartQuarter();
                Date EndRQ=GetEndQuarter();

                Instance[] absentstudents = selectInState("Student", "Absent");
                for (int i = 0; i < absentstudents.length; i++) {
                        Date MinorDate=absentstudents[i].getDate("Start of minor");
                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                        Sumofabsentdaysofabsentstudents += absentstudents[i].getInteger("Absent days"); }
                }

                Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
                for (int i = 0; i < absent_reportedstudents.length; i++) {
                        Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");
                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                        Sumofabsentdaysofabsentstudents += absent_reportedstudents[i].getInteger("Absent
days"); }
                }

                Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
                for (int i = 0; i < absent_informedstudents.length; i++) {
                        Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");
                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                        Sumofabsentdaysofabsentstudents += absent_informedstudents[i].getInteger("Absent
days"); }
                }

                return Sumofabsentdaysofabsentstudents;
                }

// Sum of students returned after mentor support

        public int getPercentageStudentsReturnedAfterMentorSupport() {

                int SumOfStudentsReturnedAfterMentorSupport = 0;
                int PercentageStudentsReturnedAfterMentorSupport = 0;
                int TotalNumberOfStudents=getTotalNumberOfStudents();

                Date StartRQ=GetStartQuarter();
                Date EndRQ=GetEndQuarter();

                Instance[] studentsreturnedaftermentorsupport = selectInState("Student", "Present");
                for (int i = 0; i < studentsreturnedaftermentorsupport.length; i++) {
                Date MinorDate=studentsreturnedaftermentorsupport[i].getDate("Start of minor");

                        if (studentsreturnedaftermentorsupport[i].getBoolean("Student returned after absence")
&&
                        (studentsreturnedaftermentorsupport[i].getBoolean("Mentor absence support") &&
                        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )))) {
                        SumOfStudentsReturnedAfterMentorSupport +=1; }
                }
                if (TotalNumberOfStudents == 0) {
                PercentageStudentsReturnedAfterMentorSupport = 0;
```

```
                    }
                    else
                    {
                    PercentageStudentsReturnedAfterMentorSupport =
((100)*(SumOfStudentsReturnedAfterMentorSupport) / (TotalNumberOfStudents));
                    }
                    return PercentageStudentsReturnedAfterMentorSupport;
                    }

// Sum of students dropped out after mentor support

          public int getPercentageStudentsDroppedOutAfterMentorSupport() {

                    int Sumofstudentsdroppedoutaftermentorsupport = 0;
                    int PercentageStudentsDroppedOutAfterMentorSupport = 0;
                    int TotalNumberOfStudents=getTotalNumberOfStudents();

                    Date StartRQ=GetStartQuarter();
                    Date EndRQ=GetEndQuarter();

                    Instance[] studentsdroppedoutaftermentorsupport = selectInState("Student", "Dropped_out");
                    for (int i = 0; i < studentsdroppedoutaftermentorsupport.length; i++) {
                    Date MinorDate=studentsdroppedoutaftermentorsupport[i].getDate("Start of minor");

                              if (studentsdroppedoutaftermentorsupport[i].getBoolean("Mentor absence support") &&
                              (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )))          {
                              Sumofstudentsdroppedoutaftermentorsupport +=1; }
                    }
                    if (TotalNumberOfStudents == 0) {
                    PercentageStudentsDroppedOutAfterMentorSupport = 0;
                    }
                    else
                    {
                    PercentageStudentsDroppedOutAfterMentorSupport =
((100)*(Sumofstudentsdroppedoutaftermentorsupport) / (TotalNumberOfStudents));
                    }
                    return PercentageStudentsDroppedOutAfterMentorSupport;
                    }

// Percentage of absent students with mentor support

public int getPercentageAbsentStudentsWithMentorSupport() {

                    int SumOfAbsentStudentsWithMentorSupport = 0;
                    int TotalNumberOfAbsentStudents=getSumOfAbsentStudents();
                    int PercentageAbsentStudentsWithMentorSupport = 0;

                    Date StartRQ=GetStartQuarter();
                    Date EndRQ=GetEndQuarter();

                    Instance[] absentstudents = selectInState("Student", "Absent");
                    for (int i = 0; i < absentstudents.length; i++) {
                    Date MinorDate=absentstudents[i].getDate("Start of minor");

                              if (absentstudents[i].getBoolean("Mentor absence support") &&
                              (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
                              SumOfAbsentStudentsWithMentorSupport +=1; }
                    }

                    Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
                    for (int i = 0; i < absent_reportedstudents.length; i++) {
                    Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");
```

```
                        if (absent_reportedstudents[i].getBoolean("Mentor absence support") &&
                        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
                        SumOfAbsentStudentsWithMentorSupport +=1; }
                }

                Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
                for (int i = 0; i < absent_informedstudents.length; i++) {
                Date MinorDate=absent_informedstudents[i].getDate("Start of minor");

                        if (absent_informedstudents[i].getBoolean("Mentor absence support") &&
                        (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ))) {
                        SumOfAbsentStudentsWithMentorSupport +=1; }
                }

                if (TotalNumberOfAbsentStudents == 0) {
                PercentageAbsentStudentsWithMentorSupport = 0;
                }
                else
                {
                PercentageAbsentStudentsWithMentorSupport =
((100)*(SumOfAbsentStudentsWithMentorSupport) / (TotalNumberOfAbsentStudents));
                }
                return PercentageAbsentStudentsWithMentorSupport;
                }

// Average number of absent days of absent students

public int getAverageNumberOfAbsentDaysOfAbsentStudents() {

                int Totalnumberofabsentstudents = 0;
                int AverageNumberOfAbsentDaysOfAbsentStudents = 0;
                int Sumofabsentdaysofabsentstudents = 0;

                Date StartRQ=GetStartQuarter();
                Date EndRQ=GetEndQuarter();

                Instance[] absentstudents = selectInState("Student", "Absent");
                for (int i = 0; i < absentstudents.length; i++) {
                Date MinorDate=absentstudents[i].getDate("Start of minor");

                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                        Sumofabsentdaysofabsentstudents += absentstudents[i].getInteger("Absent days");
                        Totalnumberofabsentstudents += 1; }
                }

                Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
                for (int i = 0; i < absent_reportedstudents.length; i++) {
                Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");

                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {

                        Sumofabsentdaysofabsentstudents += absent_reportedstudents[i].getInteger("Absent
days");
                        Totalnumberofabsentstudents += 1; }
                }

                Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
                for (int i = 0; i < absent_informedstudents.length; i++) {
                Date MinorDate=absent_informedstudents[i].getDate("Start of minor");

                        if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
```

```
                    Sumofabsentdaysofabsentstudents += absent_informedstudents[i].getInteger("Absent
days");
                    Totalnumberofabsentstudents += 1; }
                }

                if (Totalnumberofabsentstudents == 0) {
                AverageNumberOfAbsentDaysOfAbsentStudents = 0;
                }
                else
                {
                AverageNumberOfAbsentDaysOfAbsentStudents = Sumofabsentdaysofabsentstudents /
Totalnumberofabsentstudents;
                }
                return AverageNumberOfAbsentDaysOfAbsentStudents;
                }

// Percentage absent students

public int getPercentageAbsentStudents() {

                int PercentageAbsentStudents = 0;
                int Totalnumberofstudents = 0;
                int Totalnumberofabsentstudents = 0;

                Date StartRQ=GetStartQuarter();
                Date EndRQ=GetEndQuarter();

                Instance[] students = selectInState("Student", "@any");
                for (int i = 0; i < students.length; i++) {
                Date MinorDate=students[i].getDate("Start of minor");

                    if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                    Totalnumberofstudents += 1; }
                }

                Instance[] absentstudents = selectInState("Student", "Absent");
                for (int i = 0; i < absentstudents.length; i++) {
                Date MinorDate=absentstudents[i].getDate("Start of minor");

                    if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {

                    Totalnumberofabsentstudents += 1; }
                }

                Instance[] absent_reportedstudents = selectInState("Student", "Absent_Reported");
                for (int i = 0; i < absent_reportedstudents.length; i++) {
                Date MinorDate=absent_reportedstudents[i].getDate("Start of minor");

                    if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {

                    Totalnumberofabsentstudents += 1; }
                }

                Instance[] absent_informedstudents = selectInState("Student", "Absent_Reported_Informed");
                for (int i = 0; i < absent_informedstudents.length; i++) {
                Date MinorDate=absent_informedstudents[i].getDate("Start of minor");

                    if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                    Totalnumberofabsentstudents += 1; }
                }

                if (Totalnumberofabsentstudents == 0) {
                PercentageAbsentStudents = 0;
```

```
                    }
                    else
                    {
                    PercentageAbsentStudents = ((100)*(Totalnumberofabsentstudents) / (Totalnumberofstudents));
                    }
                    return PercentageAbsentStudents;
                    }

// Percentage absent students reported in time

public int getPercentageStudentsReportedInTime() {

                    int PercentageStudentsReportedInTime = 0;
                    int AbsentReportedReportedWithinDays = 0;
                    int AbsentReportedInformedReportedWithinDays = 0;
                    int TotalShouldBeReportedStudents = 0;
                    int ReportedWithinTime = 0;
                    int NotPresentDays = 0;
                    int AbsentDays = 0;

                    Date StartRQ=GetStartQuarter();
                    Date EndRQ=GetEndQuarter();

                    Instance[] reportedstudents = selectInState("Student", "Absent_Reported");
                    for (int i = 0; i < reportedstudents.length; i++) {
                    Date MinorDate=reportedstudents[i].getDate("Start of minor");
                    AbsentReportedReportedWithinDays = reportedstudents[i].getInteger("Reported within days");

                              if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentReportedReportedWithinDays <= 5)) {
                              TotalShouldBeReportedStudents += 1;
                              ReportedWithinTime += 1; }

                              if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentReportedReportedWithinDays > 5)) {
                              TotalShouldBeReportedStudents += 1; }

                    }

                    Instance[] reportedinformedstudents = selectInState("Student", "Absent_Reported_Informed");
                    for (int i = 0; i < reportedinformedstudents.length; i++) {
                    Date MinorDate=reportedinformedstudents[i].getDate("Start of minor");
                    AbsentReportedInformedReportedWithinDays = reportedinformedstudents[i].getInteger("Reported
within days");

                              if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentReportedInformedReportedWithinDays <= 5)) {
                              TotalShouldBeReportedStudents += 1;
                              ReportedWithinTime += 1;
                              }

                              if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentReportedInformedReportedWithinDays > 5)) {
                              TotalShouldBeReportedStudents += 1;
                              }
                    }

                    Instance[] notpresentstudents = selectInState("Student", "Not_present");
                    for (int i = 0; i < notpresentstudents.length; i++) {
                    Date MinorDate=notpresentstudents[i].getDate("Start of minor");
                    NotPresentDays = notpresentstudents[i].getInteger("Not present days");
                              if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(NotPresentDays >= 20)) {
```

```
                                TotalShouldBeReportedStudents += 1;
                                }
                        }

                        Instance[] absentstudents = selectInState("Student", "Absent");
                        for (int i = 0; i < absentstudents.length; i++) {
                        Date MinorDate=absentstudents[i].getDate("Start of minor");
                        AbsentDays = absentstudents[i].getInteger("Absent days");
                                if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentDays > 5)) {
                                TotalShouldBeReportedStudents += 1;
                                }
                        }

                        if (TotalShouldBeReportedStudents == 0) {
                        PercentageStudentsReportedInTime = 0; }

                        else
                        {
                        PercentageStudentsReportedInTime = ((100)*(ReportedWithinTime) /
(TotalShouldBeReportedStudents));

                        }
                        return PercentageStudentsReportedInTime;
                        }

// Sum of students/parents need to be informed

                        public int getSumOfStudentsNeedToBeInformed() {

                        int SumOfStudentsNeedToBeInformed = 0;

                        Date StartRQ=GetStartQuarter();
                        Date EndRQ=GetEndQuarter();

                        Instance[] reportedstudents = selectInState("Student", "Absent_Reported");
                        for (int i = 0; i < reportedstudents.length; i++) {
                                Date MinorDate=reportedstudents[i].getDate("Start of minor");
                                if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 )) {
                                SumOfStudentsNeedToBeInformed += 1; }
                        }
                        return SumOfStudentsNeedToBeInformed;
                        }

// Sum of students need to be reported

                        public int getSumOfStudentsNeedToBeReported() {

                        int SumOfStudentsNeedToBeReported = 0;
                        int NotPresentDays = 0;
                        int AbsentDays = 0;

                        Date StartRQ=GetStartQuarter();
                        Date EndRQ=GetEndQuarter();

                        Instance[] notpresentstudents = selectInState("Student", "Not_present");
                        for (int i = 0; i < notpresentstudents.length; i++) {
                        Date MinorDate=notpresentstudents[i].getDate("Start of minor");
                        NotPresentDays = notpresentstudents[i].getInteger("Not present days");
                                if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(NotPresentDays >= 15)) {
                                SumOfStudentsNeedToBeReported += 1;
                                }
```

```
                        }

                        Instance[] absentstudents = selectInState("Student", "Absent");
                        for (int i = 0; i < absentstudents.length; i++) {
                        Date MinorDate=absentstudents[i].getDate("Start of minor");
                        AbsentDays = absentstudents[i].getInteger("Absent days");
                                if (MinorDate.compareTo(StartRQ)>=0 && (MinorDate.compareTo(EndRQ)<0 ) &&
(AbsentDays > 5)) {
                                SumOfStudentsNeedToBeReported += 1;
                                }
                        }
                        return SumOfStudentsNeedToBeReported;
                        }
}
```

DashboardInit.java

```
package ROCvA_Verzuimproces;;

import com.metamaxim.modelscope.callbacks.*;

public class DashboardInit {

        public static String defaultDashboardName() {
          return "KPIs Absentee Process";
        }

}
```

InformControl.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class InformControl extends Behaviour {

        public String getState() {
                        if (this.getInteger("Student age") < 18) return "Minor";
                        else return "Not_minor";
                }
}
```

InformStudent.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class InformStudent extends Event {

        public void handleEvent() {

                this.submitToModel();

                Instance aStudent = this.getInstance("Student");

                Event Inform = this.createEvent("Inform");
                Inform.setInstance("Student", aStudent);
```

```
                Inform.submitToModel();
        }

}
```

InformStudentAndParents.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class InformStudentAndParents extends Event {

        public void handleEvent() {

                this.submitToModel();

                Instance aStudent = this.getInstance("Student");

                Event Inform = this.createEvent("Inform");
                Inform.setInstance("Student", aStudent);

                Inform.submitToModel();
        }

}
```

MarkAsNotPresent.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class MarkAsNotPresent extends Event {

        public void setNotPresentSinceDate(EventValueAttribute attribute, Instance selected, String subscript) {
        attribute.setDate(MarkAsNotPresentInit.defaultNotPresentSinceDate());
        }

}
```

MarkAsNotPresentInit.java

```
package ROCvA_Verzuimproces;;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class MarkAsNotPresentInit {

        public static Date defaultNotPresentSinceDate() {
          Date today = new Date();
          return today;
        }

}
```

ReportStudent.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class ReportStudent extends Event {

	public void handleEvent() {

		this.submitToModel();

		Instance aStudent = this.getInstance("Student");

		Event Report = this.createEvent("Report");
		Report.setInstance("Student", aStudent);

		Report.submitToModel();
	}

}
```

Student.java

```
package ROCvA_Verzuimproces;

import com.metamaxim.modelscope.callbacks.*;
import java.util.*;

public class Student extends Behaviour {

public String getCurrentState() {
		return this.getState("Student");
		}

public int getStudentAge() {
   Calendar today = Calendar.getInstance();
	Calendar birthDate = Calendar.getInstance();

   int age = 0;

	Date bd=this.getDate("Student birthday");
   birthDate.setTime(bd);

   age = today.get(Calendar.YEAR) - birthDate.get(Calendar.YEAR);

   if		(birthDate.get(Calendar.MONTH) > today.get(Calendar.MONTH ))
			{ age--; }

   else if ((birthDate.get(Calendar.MONTH) == today.get(Calendar.MONTH )) &&
        (birthDate.get(Calendar.DAY_OF_MONTH) > today.get(Calendar.DAY_OF_MONTH )))
			{ age--; }

	return age;
                                                                        }

public int getNotPresentDays() {

	int notPresentDays;

	String cs=this.getString("Current state");
```

```
	if (cs.equals("Not_present")) {

		long MSPERDAY = 60 * 60 * 24 * 1000;

		Calendar dateStartCal = Calendar.getInstance();
		Date nd=this.getDate("Not present since date");
		dateStartCal.setTime(nd);

		Calendar dateEndCal = Calendar.getInstance();

		long dateDifferenceInDays = ( dateEndCal.getTimeInMillis() - dateStartCal.getTimeInMillis() ) /
MSPERDAY;

		notPresentDays = (int) dateDifferenceInDays; }

		else {
		notPresentDays = 0; }

		return notPresentDays;
                                                                                }

public int getAbsentDays() {

	int AbsentDays;

	String cs=this.getString("Current state");

	if (cs.equals("Absent") || cs.equals("Absent_Reported") || (cs.equals("Absent_Reported_Informed"))) {

		long MSPERDAY = 60 * 60 * 24 * 1000;

		Calendar dateStartCal = Calendar.getInstance();
		Date ad=this.getDate("Not present since date");
		dateStartCal.setTime(ad);

		Calendar dateEndCal = Calendar.getInstance();

		long dateDifferenceInDays = ( dateEndCal.getTimeInMillis() - dateStartCal.getTimeInMillis() ) /
MSPERDAY;

		AbsentDays = (int) dateDifferenceInDays;
		AbsentDays = AbsentDays - 14; }

		else {
		AbsentDays = 0; }

		return AbsentDays;
                                                                                }

public int getReportedWithinDays() {

	int ReportedWithinDays = 0;

	String cs=this.getString("Current state");

	if (cs.equals("Absent_Reported") || cs.equals("Absent_Reported_Informed")) {

		long MSPERDAY = 60 * 60 * 24 * 1000;

		Calendar dateStartCal = Calendar.getInstance();
		Date nd=this.getDate("Not present since date");
		dateStartCal.setTime(nd);
```

```
                    Calendar dateEndCal = Calendar.getInstance();
                    Date rd=this.getDate("Reported since date");
                    dateEndCal.setTime(rd);

                    long dateDifferenceInDays = ( dateEndCal.getTimeInMillis() - dateStartCal.getTimeInMillis() ) /
MSPERDAY;

                    ReportedWithinDays = (int) dateDifferenceInDays;
                    ReportedWithinDays = ReportedWithinDays - 14; }

                    else {
                    ReportedWithinDays = 0; }

                    return ReportedWithinDays;
                                                                    }
                                                                                                    }
```

Appendix C: Artificial test data

```
# Metamaxim ModelScope Instances File written on Wed Dec 20 11:56:35 CET 2017

INSTANCE : AbsenteeWorker_Richard = 1
        BEHAVIOUR : AbsenteeWorker = Created
        AbsenteeWorker Name : String = AbsenteeWorker_Richard

INSTANCE : AbsenteeWorker_John = 2
        BEHAVIOUR : AbsenteeWorker = Created
        AbsenteeWorker Name : String = AbsenteeWorker_John

INSTANCE : KPIs Absentee Process = 3
        BEHAVIOUR : Dashboard = Created
        Dashboard Name : String = KPIs Absentee Process
        Start of reporting quarter : Date = 1 Aug 2017

INSTANCE : Mentor_Sjaak = 4
        BEHAVIOUR : Mentor = Created
        Mentor Name : String = Mentor_Sjaak
        Student : Student = 17

INSTANCE : Mentor_Dennis = 5
        BEHAVIOUR : Mentor = Created
        Mentor Name : String = Mentor_Dennis
        Student : Student = 16

INSTANCE : Mentor_Pascal = 6
        BEHAVIOUR : Mentor = Created
        Mentor Name : String = Mentor_Pascal
        Student : Student = 21

INSTANCE : Student_Jerry = 7
        BEHAVIOUR : Student = Not_present
        Student Name : String = Student_Jerry
        Start of minor : Date = 1 Oct 2017
        Student birthday : Date = 25 Dec 2000
        Mentor : Mentor = 4
        Teacher : Teacher = 22
        Absentee worker : AbsenteeWorker = null
        Mentor absence support : Boolean = true
        Student returned after absence : Boolean = true
        Reported since date : Date = 29 Oct 2017
        BEHAVIOUR : StudentAttended = @new
        Not present since date : Date = 22 Oct 2017
```

BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Not_allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 25 Oct 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Andreas = 8
BEHAVIOUR : Student = Dropped_out
Student Name : String = Student_Andreas
Start of minor : Date = 1 Aug 2017
Student birthday : Date = 13 Jun 1985
Mentor : Mentor = 4
Teacher : Teacher = null
Absentee worker : AbsenteeWorker = null
Mentor absence support : Boolean = true
Student returned after absence : Boolean = true
Reported since date : Date = 29 Oct 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 13 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 26 Oct 2017
BEHAVIOUR : InformControl = Not_minor

INSTANCE : Student_Bob = 9
BEHAVIOUR : Student = Absent_Reported
Student Name : String = Student_Bob
Start of minor : Date = 27 Oct 2017
Student birthday : Date = 5 Mar 2000
Mentor : Mentor = 4
Teacher : Teacher = 23
Absentee worker : AbsenteeWorker = 1
Mentor absence support : Boolean = true
Student returned after absence : Boolean = true
Reported since date : Date = 13 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 19 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 26 Oct 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Erik = 10
BEHAVIOUR : Student = Absent_Reported
Student Name : String = Student_Erik
Start of minor : Date = 27 Oct 2017
Student birthday : Date = 26 Dec 2000
Mentor : Mentor = 5
Teacher : Teacher = 23
Absentee worker : AbsenteeWorker = 2
Mentor absence support : Boolean = true
Student returned after absence : Boolean = true
Reported since date : Date = 14 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 30 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 14 Nov 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Jeroen = 11
BEHAVIOUR : Student = Not_present
Student Name : String = Student_Jeroen
Start of minor : Date = 27 Oct 2017

Student birthday : Date = 26 Oct 1999
Mentor : Mentor = 5
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = null
Mentor absence support : Boolean = false
Student returned after absence : Boolean = true
Reported since date : Date = 4 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 22 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Not_allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 4 Nov 2017
BEHAVIOUR : InformControl = Not_minor

INSTANCE : Student_Sander = 12
BEHAVIOUR : Student = Absent_Reported
Student Name : String = Student_Sander
Start of minor : Date = 27 Oct 2017
Student birthday : Date = 26 Oct 1999
Mentor : Mentor = 5
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = 1
Mentor absence support : Boolean = true
Student returned after absence : Boolean = false
Reported since date : Date = 11 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 26 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 26 Oct 2017
BEHAVIOUR : InformControl = Not_minor

INSTANCE : Student_Frits = 13
BEHAVIOUR : Student = Absent_Reported
Student Name : String = Student_Frits
Start of minor : Date = 1 Oct 2017
Student birthday : Date = 28 Oct 2001
Mentor : Mentor = 5
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = 2
Mentor absence support : Boolean = true
Student returned after absence : Boolean = true
Reported since date : Date = 13 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 26 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 28 Oct 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Maarten = 14
BEHAVIOUR : Student = Present
Student Name : String = Student_Maarten
Start of minor : Date = 1 Oct 2017
Student birthday : Date = 28 Oct 1999
Mentor : Mentor = 5
Teacher : Teacher = null
Absentee worker : AbsenteeWorker = null
Mentor absence support : Boolean = false
Student returned after absence : Boolean = false
Reported since date : Date = 29 Oct 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 28 Oct 2017

```
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
        Informed date : Date = 28 Oct 2017
        BEHAVIOUR : InformControl = Not_minor

INSTANCE : Student_Dirk = 15
        BEHAVIOUR : Student = Absent_Reported
        Student Name : String = Student_Dirk
        Start of minor : Date = 1 Oct 2017
        Student birthday : Date = 28 Jan 2001
        Mentor : Mentor = 5
        Teacher : Teacher = 22
        Absentee worker : AbsenteeWorker = 2
        Mentor absence support : Boolean = false
        Student returned after absence : Boolean = false
        Reported since date : Date = 13 Nov 2017
        BEHAVIOUR : StudentAttended = @new
        Not present since date : Date = 28 Oct 2017
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
        Informed date : Date = 28 Oct 2017
        BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Bart = 16
        BEHAVIOUR : Student = Present
        Student Name : String = Student_Bart
        Start of minor : Date = 1 Oct 2017
        Student birthday : Date = 16 Nov 2000
        Mentor : Mentor = 5
        Teacher : Teacher = 22
        Absentee worker : AbsenteeWorker = 1
        Mentor absence support : Boolean = true
        Student returned after absence : Boolean = true
        Reported since date : Date = 13 Nov 2017
        BEHAVIOUR : StudentAttended = @new
        Not present since date : Date = 13 Oct 2017
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
        Informed date : Date = 28 Oct 2017
        BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Henk = 17
        BEHAVIOUR : Student = Present
        Student Name : String = Student_Henk
        Start of minor : Date = 1 Sep 2017
        Student birthday : Date = 10 Jul 2002
        Mentor : Mentor = 4
        Teacher : Teacher = 22
        Absentee worker : AbsenteeWorker = 1
        Mentor absence support : Boolean = true
        Student returned after absence : Boolean = true
        Reported since date : Date = 29 Oct 2017
        BEHAVIOUR : StudentAttended = @new
        Not present since date : Date = 31 Oct 2017
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
        Informed date : Date = 28 Oct 2017
        BEHAVIOUR : InformControl = Minor

INSTANCE : Student_David = 18
        BEHAVIOUR : Student = Absent_Reported
        Student Name : String = Student_David
        Start of minor : Date = 1 Oct 2017
```

Student birthday : Date = 7 Jul 2000
Mentor : Mentor = 5
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = 1
Mentor absence support : Boolean = false
Student returned after absence : Boolean = true
Reported since date : Date = 13 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 23 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 29 Oct 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Joop = 19
BEHAVIOUR : Student = Present
Student Name : String = Student_Joop
Start of minor : Date = 1 Oct 2017
Student birthday : Date = 20 Feb 1999
Mentor : Mentor = 5
Teacher : Teacher = null
Absentee worker : AbsenteeWorker = null
Mentor absence support : Boolean = false
Student returned after absence : Boolean = false
Reported since date : Date = 29 Oct 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 29 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 29 Oct 2017
BEHAVIOUR : InformControl = Not_minor

INSTANCE : Student_Lucas = 20
BEHAVIOUR : Student = Not_present
Student Name : String = Student_Lucas
Start of minor : Date = 1 Aug 2017
Student birthday : Date = 24 Apr 2001
Mentor : Mentor = 6
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = null
Mentor absence support : Boolean = false
Student returned after absence : Boolean = false
Reported since date : Date = 1 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 17 Oct 2017
BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Not_allowed
BEHAVIOUR : StudentInformed = @new
Informed date : Date = 1 Nov 2017
BEHAVIOUR : InformControl = Minor

INSTANCE : Student_Daan = 21
BEHAVIOUR : Student = Present
Student Name : String = Student_Daan
Start of minor : Date = 1 Sep 2017
Student birthday : Date = 1 Jan 2000
Mentor : Mentor = 6
Teacher : Teacher = 22
Absentee worker : AbsenteeWorker = 1
Mentor absence support : Boolean = true
Student returned after absence : Boolean = true
Reported since date : Date = 15 Nov 2017
BEHAVIOUR : StudentAttended = @new
Not present since date : Date = 31 Oct 2017

```
        BEHAVIOUR : AllowedNumberofNotPresentDaysControl = Allowed
        BEHAVIOUR : StudentInformed = @new
        Informed date : Date = 15 Nov 2017
        BEHAVIOUR : InformControl = Minor

INSTANCE : Teacher_Michael = 22
        BEHAVIOUR : Teacher = Created
        Teacher Name : String = Teacher_Michael

INSTANCE : Teacher_Onno = 23
        BEHAVIOUR : Teacher = Created
        Teacher Name : String = Teacher_Onno
```